UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
Machine and Deep Learning in Data-Driven Structural Health Monitoring

Permalink
https://escholarship.org/uc/item/45q0z1mp

Author
Gao, Yuqing

Publication Date
2020

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

Machine and Deep Learning in Data-Driven Structural Health Monitoring

by

Yuqing Gao

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Civil and Environmental Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Khalid M. Mosalam, Chair
Professor Jack P. Moehle
Professor Aditya Guntuboyina

Spring 2020

Machine and Deep Learning in Data-Driven Structural Health Monitoring

Copyright 2020
by
Yuqing Gao

Abstract

Machine and Deep Learning in Data-Driven Structural Health Monitoring

by

Yuqing Gao

Doctor of Philosophy in Engineering – Civil and Environmental Engineering

University of California, Berkeley

Professor Khalid M. Mosalam, Chair

In this data explosion epoch, data-driven structural health monitoring (SHM) and rapid damage assessment after natural hazards have become of great interest in civil engineering. In this dissertation, the state-of-the-art machine learning (ML) and deep learning (DL) technologies are introduced and implemented for vision-based and vibration-based SHM applications, respectively. Specifically, the dissertation consists of five core parts, namely (1) motivation, background and basics of ML/DL-based data-driven SHM, (2) general topics of vision-based SHM, (3) advanced topics of vision-based SHM, (4) ML in vibration-based SHM, and (5) conclusions and extensions.

Using DL in vision-based SHM is a relatively new research direction in civil and structural engineering. As researchers begin to apply these concepts to the structural engineering domain, several critical issues need to be addressed: (1) the lack of uniform automated response and damage detection principles based on domain knowledge, (2) the lack of benchmark datasets with well-labeled large amounts of data, and (3) the lack of interpretation of the trained ML/DL models. To address the first two issues, an automated and hierarchical framework has been proposed and developed herein: the ϕ-Net for the PEER[1] Hub Image-Net. This framework consists of eight basic benchmark detection tasks based on current domain knowledge and past reconnaissance experience, namely (1) scene level, (2) damage state, (3) concrete cover spalling condition (material loss), (4) material type, (5) collapse mode, (6) component type, (7) damage level, and (8) damage type. According to the ϕ-Net framework, a large number of structural images was collected, preprocessed, and labeled to form an open-source online large-scale multi-attribute image dataset, namely, the ϕ-Net dataset, which currently contains 36,413 images with multiple labels. Before conducting comprehensive numerical experiments on the ϕ-Net dataset, a preliminary study was performed firstly on a smaller scale dataset for validating the effectiveness and applicability of the DL methods in vision-based SHM along with the transfer learning (TL) technique. In the study of the

[1]Pacific Earthquake Engineering Research (PEER) Center.

TL, instead of training a network from scratch, two different strategies, namely feature extractor and fine-tuning with pre-trained models, e.g., VGGNet (Visual Geometry Group) and Residual Network (ResNet), were pursued. A series of computer experiments were designed based on the properties of these two strategies to study the relative optimal model setting and scope of application. With a preliminary sense from the TL experiments, a comprehensive investigation on applying DL in general vision tasks in the ϕ-Net was made through comparative experiments with various DL models and training strategies, i.e., TL and data augmentation (DA). Promising results were achieved and reported, which provide the reference for future DL applications. Moreover, two direct applications of the ϕ-Net were further performed, namely image-based post-disaster assessment of the 1999 Chi-Chi earthquake and the 2018 ϕ-Net Challenge, which revealed the great potential and contribution of the ϕ-Net in vision-based SHM. To address the "black box" issue implying that the working principle of DL is hard to understand by humans, several state-of-the-art visual interpretation methods were applied in explaining the DL models in SHM problems. Beyond the general view over the basic (non-expert) structural vision tasks, i.e., spalling condition task, several key factors including learning procedure, learned features from different network depths, training techniques, and level of semantic abstraction were effectively explored.

As one advanced technology in DL, Generative Adversarial Network (GAN) was investigated to overcome the shortcomings of conventional DA, alleviate data deficit, and finally improve the model performance under certain conditions. Unlike normal natural objects, the distribution of structural images is much more complex, thus, Deep Convolutional GAN (DCGAN) along with Leaf-Bootstrapping (LB) training method was proposed to improve the quality of synthetic images. To effectively and quantitatively evaluate the quality of the synthetic images generated by DCGAN to complement human inspection, Self-Inception Score (SIS) and Generalization Ability (GA) were proposed. Four experimental cases were designed to explore the performance of LB based GAN under different situations with the above-mentioned metrics. To further explore the potentials of using synthetic images to improve the classifier performance, a well-designed experiment was conducted under the constraints of limited computational resources and low-data regime, where a union training strategy, namely synthetic data fine-tuning (SDF), was proposed. Moreover, to improve the efficiency of GAN-based augmentation method and also to take the imbalanced-label issue into account, a semi-supervised learning GAN pipeline along with the balanced batch sampling technique was proposed, and named Balanced Semi-supervised GAN (BSS-GAN). A series of complementary experiments with respect to the detection of the reinforced concrete cover spalling was designed and conducted, where comparisons were made among conventional training pipeline, directly synthetic data aggregation and BSS-GAN. The experimental results demonstrated the effectiveness and robustness of these proposed methods.

Besides the vision-based SHM, the effectiveness of ML in vibration-based SHM was also explored while combining ML with the time series (TS) approaches. Autoregressive (AR) series model, as one type of the TS methods, is useful for structural damage detection in

the area of vibration-based SHM. However, due to several limitations (e.g. non-stationary conditions and subjective feature selection), it did not gain much attention during the past few years. With the boosting trends of ML technologies, this study combined TS modeling and ML classification together to automatically select the features and overcome several limitations. First, an automated end-to-end detection framework/algorithm was proposed, namely auto-regressive integrated moving average–machine learning (ARIMA-ML) with modules for smoothing-segmentation-normalization-differencing (SSND), auto-stationarity check, ARIMA feature extraction, and ML classification. In addition, an ensemble method using voting mechanism was adopted in the ML classification module. Based on shaking table tests of a steel building frame, floor acceleration data were collected and labeled according to experimental observations and records. Subsequently, three damage detection tasks were designed for: (1) general damage state, (2) local damage state, and (3) local damage pattern, where the influence of overlapping in the segmentation sliding windows was considered. In addition, to better understand the ML models, the importance of the features was also discussed along with the classification results. Moreover, the results of numerical experiments indicated the robustness and promising performance of the proposed algorithm in all considered tasks, which is expected to be useful for future damage assessment of instrumented structures by structural engineers during rapid reconnaissance efforts after natural hazard events, e.g., earthquakes.

To those who bring passion and love to my life.

Contents

Contents **ii**

List of Figures **v**

List of Tables **ix**

1 Introduction **1**
1.1 Background & Motivations 1
1.2 Related Work 7
1.3 Objectives & Contributions 14
1.4 Organization of the Dissertation 19

2 Basics of Machine Learning **21**
2.1 Supervised Learning 21
2.2 Logistic Regression 27
2.3 Support Vector Machine 30
2.4 Decision Tree & Random Forest 36

3 Basics of Deep Learning **41**
3.1 Neural Network 41
3.2 Convolutional Neural Network (CNN) 47
3.3 Deep Convolutional Neural Network 53

4 PEER Hub ImageNet *(ϕ-Net)* **60**
4.1 Automatic Detection Framework of the ϕ-Net 60
4.2 Data Preparation 72
4.3 Data Summary and Extension 76
4.4 Extensions of the Dataset Hierarchy 80

5 TL Using a Small Structural ImageNet **83**
5.1 Transfer Learning (TL) 83
5.2 Experimental Objectives 87
5.3 Experimental Setup 89

5.4 Experimental Results & Discussions 97
5.5 Summary 108

6 Benchmark Experiments on the ϕ-Net **110**
6.1 Experimental Objectives 110
6.2 Experimental Setup 111
6.3 Experimental Results & Discussions 112
6.4 Summary 124

7 Applications & Extensions of the ϕ-Net **130**
7.1 Post-disaster Damage Assessment 130
7.2 2018 ϕ-Net Challenge 132

8 Visual Interpretation Methods of Deep CNN **143**
8.1 Guided-backpropagation (GBP) 144
8.2 Class Activation Map (CAM) 145
8.3 Gradient-weighted CAM (Grad-CAM) 146
8.4 Guided Grad-CAM 147
8.5 Local Interpretable Model-Agnostic Explanations (LIME) 147

9 Interpreting CNN in Structural Vision Tasks **149**
9.1 Preliminary Interpretation of the Structural Vision Tasks 149
9.2 Experimental Objectives 151
9.3 Experimental Setup 154
9.4 Experimental Results & Discussions 158
9.5 Summary 170

10 GAN for Image Generation **173**
10.1 Principles & Mathematical Formulation of GAN 174
10.2 Deep Convolutional GAN (DCGAN) 176
10.3 Evaluation Metrics 178

11 GAN for Structural Image DA & Classification **182**
11.1 Leaf-Bootstrapping Augmentation 182
11.2 Experiments on Synthetic Structural Image Generation 185
11.3 Experiments on Synthetic Data Augmentation for Classification 197
11.4 Summary 201

12 BSS-GAN for Structural Image Classification **203**
12.1 Formulation of Balanced Semi-supervised GAN (BSS-GAN) 203
12.2 Experimental Objectives 205
12.3 Experimental Setup 206
12.4 Experimental Results & Discussions 208

12.5 Summary 212

13 Fusion of Time Series & Machine Learning **214**
13.1 Basics of Time Series Modeling 214
13.2 ARIMA-ML Algorithm 219
13.3 Summary 222

14 ARIMA-ML for Structural Damage Recognition **224**
14.1 Experimental Data Source 224
14.2 Experimental Objectives 226
14.3 Exploratory Data Analysis 228
14.4 Experimental Setup 234
14.5 Experimental Results & Discussions 235
14.6 Summary 244

15 Conclusions & Extensions **246**

Bibliography **255**

List of Figures

2.1 Confusion matrix 25
2.2 Illustration of appropriate fitting, over-fitting, and under-fitting 27
2.3 Example of a 5-fold cross-validation 27
2.4 Illustration of the distance D by projecting $(z - x_0)$ on w 31
2.5 Illustration of a hypothetical DT model for three-dimensional data 37
2.6 Prediction procedure of the RF 40

3.1 Example of a three-layer NN 43
3.2 Dropout operation in the NN model 47
3.3 Illustration of the Conv and pooling operations in a CNN for vision-based SHM 47
3.4 Illustration of the Conv operations in a simple example 48
3.5 Example of one unit padding on a 3×3 input 50
3.6 Example of max pooling and average pooling 51
3.7 Illustration of the two original VGGNet architectures 55
3.8 Example of one Inception module with four branches 56
3.9 Two types of shortcut connections in ResNet 58
3.10 Illustration of the ResNet-50 (49 Conv + 1 Dense) network architecture 59

4.1 Automatic detection framework of the ϕ-Net 61
4.2 Sample images used in the scene level task 63
4.3 Sample images used in the damage state task 64
4.4 Sample images used in the spalling condition task 65
4.5 Sample images used in the material type task 66
4.6 Sample images used in the collapse mode task 67
4.7 Sample images used in the component type task 69
4.8 Sample images used in the damage level task 70
4.9 Sample images used in the damage type task 71
4.10 Sub-parts are cropped from a high-resolution structural level image 73
4.11 Graphical user interface of the online labeling tool 74
4.12 Multi-attribute split algorithm 75
4.13 Statistics of the labeled data in the ϕ-Net dataset 77
4.14 Hierarchy of the extended ϕ-Net framework 81
4.15 Shallow depth representation of the ϕ-Net 82

5.1 Transfer learning flow chart used in this dissertation 85
5.2 Transfer learning configuration of VGG-16 86
5.3 Label ratios in the ϕ-Net-2000 dataset 90
5.4 Configuration of the feature extractor for network input size experiment 93
5.5 Configuration of the fine-tuning for the retrain ratio experiment 95
5.6 Accuracy history of four recognition tasks using feature extractor for the network input size experiment 99
5.7 Normalized confusion matrices of the test prediction using feature extractor for the network input size experiment 101
5.8 Loss and accuracy history of the four recognition tasks using fine-tuning for the retrain ratio experiment 103
5.9 Normalized confusion matrices of the test predictions by fine-tuning two Conv blocks from the retrain ratio experiment 105
5.10 Accuracy history of ResNet-50 and ResNet-101 using feature extractor for the network comparison experiment 107
5.11 Accuracy history of ResNet-50 and ResNet-101 using fine-tuning for the network comparison experiment 108

6.1 Training history of DL models for 50 epochs in the scene level task 113
6.2 Training history of DL models for 50 epochs in the damage state task 115
6.3 Training history of DL models for 50 epochs in the spalling condition task . . . 117
6.4 Training history of DL models for 50 epochs in the material type task 118
6.5 Training history of DL models for 50 epochs in the collapse mode task 120
6.6 Training history of DL models for 50 epochs in the component type task 122
6.7 Training history of DL models for 50 epochs in the damage level task 123
6.8 Training history of DL models for 50 epochs in the damage type task 125
6.9 Accuracy history of the best models for 50 epochs in all tasks 127

7.1 Results predicted by the SIM for the 1999 Chi-Chi earthquake images 131
7.2 Flyer for the 2018 ϕ-Net Challenge 133
7.3 Registered team distribution of the 2018 ϕ-Net Challenge 135
7.4 Homepage of Task 1 scene level identification of the 2018 ϕ-Net Challenge . . . 136
7.5 Rank of the scores in the Kaggle leaderboards for the 2018 ϕ-Net Challenge . . 138
7.6 Submissions by all teams for each task of the 2018 ϕ-Net Challenge 139
7.7 Final results of submissions by teams for each task of the 2018 ϕ-Net Challenge 140

8.1 Illustration of the forward and backward signal propagation in the CNN 144
8.2 Example of the formulation of restoring gradient in GBP 145
8.3 Illustration of generating saliency map of CAM and Grad-CAM 146
8.4 Illustration of generating saliency map of Guided Grad-CAM 147

9.1 CAM for VGGNet in four tasks 150
9.2 Failure cases with CAM questionable explanations 151

9.3 LIME for ResNet-50 in four tasks . . . 152
9.4 The confusion matrices for the adapted VGG-16 under different scenarios . . . 157
9.5 Saliency maps in predicting SP with correct predictions . . . 158
9.6 Saliency maps in predicting SP with incorrect predictions . . . 160
9.7 Two examples of CorRes results at different training epochs . . . 162
9.8 Saliency maps of two CorRes examples from different network depths . . . 164
9.9 Saliency maps of BL (IncUnr) . . . 166
9.10 Four examples of saliency maps for VGG-TL and VGG-TL-DA . . . 167
9.11 Saliency maps for FLEX . . . 168
9.12 Saliency maps for SHEAR . . . 168
9.13 Saliency maps for COMB . . . 169

10.1 Procedure of GAN training . . . 174
10.2 Illustration of the DCGAN architecture . . . 177
10.3 Split plan for computing the SIS . . . 180

11.1 Images generated by GAN trained with all the training images . . . 183
11.2 Diagram of a mixture distribution for the structural images in an abstract dimensional space . . . 184
11.3 Leaf-Bootstrapping for synthetic scene level using ϕ-Net depth-2 representation 185
11.4 Label ratio of the ϕ-Net-GAN . . . 187
11.5 Real images vs synthetic images generated from four cases . . . 191
11.6 Confusion matrices of the GA in the scene level identification . . . 194
11.7 Illustration of the synthetic data fine-tuning (SDF) pipeline . . . 198
11.8 Two learning modes of conventional training and SDF . . . 199

12.1 Trends of BSS-GAN vs BSL for increasing amounts of labeled data . . . 209
12.2 Confusion matrices of different pipelines . . . 210
12.3 Sample images from the real dataset and the synthetic dataset using the GAN-based pipelines . . . 211

13.1 Automatic end-to-end framework for the developed ARIMA-ML algorithm . . . 220
13.2 Illustration of the segmentation operation using sliding window for one sample TS signal . . . 221
13.3 Pseudo-code of the ARIMA-ML algorithm . . . 223

14.1 3-story tension-only concentrically braced beam-through frame (TCBBF) . . . 225
14.2 Sensors layout . . . 225
14.3 Loosening of the braces . . . 226
14.4 Structural idealization of the tested steel structure . . . 228
14.5 STD value, ACF and PACF of TS segment with different preprocessing . . . 230
14.6 BIC values for sub-models computed on four sample TS segments . . . 232
14.7 Residual diagnosis for three candidate models . . . 233

14.8 FI of three models for different sliding window patterns using RF 237
14.9 FI of ARI(7, 1) in local damage state detection 239
14.10 FI of ARIMA(6, 1, 6) in local damage state detection 240
14.11 FI of ARIMA(7, 1, 6) in local damage state detection 241
14.12 Confusion matrices of ARI(7, 1) with EVC for floor 2 243
14.13 FI of ARI(7, 1) model in local damage pattern recognition 243

List of Tables

4.1 Number of images in the scene level task 77
4.2 Number of images in the damage state task 78
4.3 Number of images in the spalling condition task 78
4.4 Number of images in the material type task 78
4.5 Number of images in the collapse mode task 78
4.6 Number of images in the component type task 78
4.7 Number of images in the damage level task 78
4.8 Number of images in the damage type task 79
4.9 Label distribution in training, test, and full dataset in the scene level task . . . 79
4.10 Label distribution in training, test, and full dataset in the damage state task . . 79
4.11 Label distribution in training, test, and full dataset in the spalling condition task 79
4.12 Label distribution in training, test, and full dataset in the material type task . . 79
4.13 Label distribution in training, test, and full dataset in the collapse mode task . 79
4.14 Label distribution in training, test, and full dataset in the component type task 80
4.15 Label distribution in training, test, and full dataset in the damage level task . . 80
4.16 Label distribution in training, test, and full dataset in the damage type task . . 80

5.1 Details of the network input size experiment 88
5.2 VGG-type network configuration 92
5.3 Computational time (seconds) for different network input sizes 92
5.4 Number of trainable parameters for different retrain ratios in the backpropagation 94
5.5 Computational time (seconds) for different retrain ratios 94
5.6 Configurations of ResNet-50 and ResNet-101 97
5.7 Best accuracy results (%) in the network input size experiment 100
5.8 Best accuracy results (%) in the retrain ratio experiment 104
5.9 Best test accuracy results (%) of variant VGGNet, ResNet-50 and ResNet-101 in the network comparison experiment 106

6.1 Test accuracy results (%) for each case in the scene level task 112
6.2 Test accuracy results (%) for each case in the damage state task 114
6.3 Test accuracy results (%) for each case in the spalling condition task 116
6.4 Test accuracy results (%) for each case in the material type task 116
6.5 Test accuracy results (%) for each case in the collapse mode task 119

6.6 Test accuracy results (%) for each case in the component type task 121
6.7 Test accuracy results (%) for each case in the damage level task 124
6.8 Test accuracy results (%) for each case in the damage type task 124
6.9 Summary of the best test accuracy performance (%) in all tasks 126
6.10 Average computational time (seconds) per training epoch in all tasks 129

7.1 Final results (%) of all tasks of the 2018 ϕ-Net Challenge 139
7.2 Combination weights considering task difficulty and distribution parameters . . 141
7.3 Top-3 overall performance of the 2018 ϕ-Net Challenge 142

9.1 Overall test accuracy, TPR & TNR (%) of the best adpated VGG-16 at different training epochs 156

11.1 Numbers of raw data in training and test datasets for six categories 186
11.2 Network configuration of DCGAN generator 188
11.3 Network configuration of DCGAN discriminator 188
11.4 SIS and GA for synthetic images in the scene level identification 193
11.5 SIS for synthetic images in the damage state check 195
11.6 GA for synthetic images in the damage state check 195
11.7 Validation accuracy for the experiment using synthetic data aggregation cases . 201

12.1 Label statistics of the dataset for the spalling condition classification 206
12.2 Configurations of the discriminator of the BSS-GAN and also of the BSL 207
12.3 Configuration of the generator of the BSS-GAN 207
12.4 Results (%) of BSS-GAN and BSL for varying amounts of labeled data 209

14.1 Structural information and damage patterns recorded under earthquake loading 227
14.2 Average AIC of three candidate models 231
14.3 Number of segments generated by different models (undamaged/damaged) . . . 234
14.4 Implementation details of the four ML and EVC classification algorithms 235
14.5 Accuracy of the general damage state detection for different candidate models . 235
14.6 Accuracy of the local damage state detection for different candidate models . . . 238
14.7 Accuracy of the local damage pattern recognition for different candidate models 242

Acknowledgments

No matter how time flies, it will not change the memory engraved in my heart. On the road of learning and growth, facing the failure and the setbacks are inevitable. If you fall down, you need to stand up bravely, and continue to trace the tomorrow and the dream. It is that step forward everyday, encourage myself, and sometimes wish for the miracle. Finally, I get close to my dream and award myself the highest honor in my life, that is, The Doctor of Philosophy from UC Berkeley.

On this long road of the life, thanks to who have lit the lights of my life and made me walk straight; thanks to who helped me greatly to move forward again for my dream when I fell down; thanks to who for their encouragement and support, and I feel that I am not alone. Now, at this special moment, I would like to express my sincere gratitude to those who have helped me greatly during my study at UC Berkeley.

To begin with, I wish to express my sincere appreciation to my supervisor, Professor Khalid M. Mosalam, who has the substance of a genius. He convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this study would not have been realized. If you believe in fate and coincidence, the life is destined. Prof. Mosalam is the first Berkeley professor I heard and met in the winter of 2013. Similarly, he is also the first professor I talked to when I found out about my interests in exploring new things, like artificial intelligence, and determined to pursue a Ph.D. degree. During the past 5 years, his words and his ideas inspired me and encouraged me a lot. Besides, the mentor-ship and encouragement of my study and life, he granted me with great academic freedom to explore my potentials, let alone continuous financial support for studying and attending conference. All my work is not possible without his significant support. There is an old saying in China: one day as a teacher, one life as a teacher. No matter how far I will go in the future, his mentor-ship and friendship will always be the most valued thing that accompanies me to pursue a more bright future.

I also would like to thank several professors at UC-Berkeley. In particular, Prof. Jack Moehle and Prof. Aditya Guntuboyina who served as members of my Ph.D. qualifying exam and dissertation committee, and Prof. John Canny and Prof. Matthew DeJong who also served as members of my qualifying exam. I appreciate their valuable advice on this work. Moreover, I would like to thank Prof. Filip Filippou. I lost my target and was confused with my future when I was about finishing my master study. It is his consistent and strong encouragement that made me realize what I like and what I want and finally I decided to be a Ph.D. student. In addition, special thanks are given to Prof. Wensheng Lu, Yiyi Chen, and Yue Zheng from Tongji University, who gave me the confidence and suggestions to continue my academic career after my Ph.D. study.

Besides my professors and mentors, I also greatly appreciate the assistance from my classmates, colleagues, and friends: Shuyuan Liang, Yujia Xu, Renjie Wu, Yidong Huang, Rongbing Zhou, Kuan Lu, Boyuan Kong, Jiachen Wang, Shuai Yao, Enze Cheng, Fan Hu, Peijie Li, Hagen Tam, Matthew Yeung, Lillian Lau, Victor Khong, Ke Xu, Jiawei Chen, Wen Tang, Ran Jiang, Jianfei Yang, Pengyuan Zhai, Chrystal Chern, Chenglong Li, Sachit Shroff,

Chenjie Gong, Yueshi Chen, Kevin Li, Peng Chen, Xiao Liang, Shanshan Wang, Benshun Shao, Daniel Miller, Sifat Muin, Thanh Do, Difu Liang, Jing Lian, Yue Zhang, Yajie Bao, Rui Ma, and many others I crossed path with at UC Berkeley. I would like to thank the Pacific Earthquake Engineering Research (PEER) Center staff members (Dr. Amaranth Kasalanati, Dr. Selim Günay, Mr. Gabriel Vargas, Ms. Grace Kang, Ms. Erika Donald, and Ms. Claire Johnson) and all the experts who participated in the 2018 ϕ-Net Challenge. I also would like to thank the Tsinghua-Berkeley Shenzhen Institute (TBSI) that provided me with the funding and opportunity to conduct my research and also visit and teach in Tsinghua University campus in Shenzhen.

Last and most important, I wish to express my warmest and deepest gratitude to my beloved parents and my grandfather and grandmother. I have a happy and caring family, and family members are supporting each other. It is their love and support that made me who I am today. I owe them 5-year's accompany. Being away from them that long makes me appreciate more our time together. I love you and miss you so much.

Now, I can bravely say to everyone that I proudly have a Ph.D. from the University of California, Berkeley. I am looking forward to more colorful tomorrow, and to be able to change the world to make it a better place for generations to come.

Chapter 1

Introduction

1.1 Background & Motivations

During the life-cycle of a structure, it typically goes through different types of loading stages, from the common daily loads to severe destructive external loads caused by earthquakes, tsunami, hurricanes, etc. These external loads introduce different degrees of structural damage that may lead to losses, injuries, and even fatalities. Thus, structural health monitoring (SHM) in the form of rapid and automated damage assessment after natural hazards has become an important focus in the field of structural engineering. Regarding the history of SHM, a variety of approaches and damage criteria were developed through traditional specimen testing or pattern analysis from data, where the latter one is known as data-driven SHM. The boosting of computer hardware, the increasing capacity and decreasing cost of data storage make the data-driven SHM attract increasing attention from the structural engineering community. Based on the data type and source, current data-driven SHM can be mainly divided into two directions: (1) vision-based SHM and (2) vibration-based SHM, where the vision-based SHM relies on human visual inspection in the field or from collected data of images and videos, and vibration-based SHM exploits the vibration signals from deployed sensors. However, until recently, both directions have been limited to low efficiency of either human inspection by engineers of the visual damage patterns or manually signal processing to determine the damage-related features and the level of damage that had occurred. Therefore, the current focus of data-driven SHM aims to augment and eventually replace human engineering works by automated recognition systems using data based on artificial intelligence (AI) technology.

Simply stated, AI technology is defined as algorithms and methods that can learn and simulate the behavior of human beings and then realize automation for certain tasks. Nowadays, machine learning (ML) and deep learning (DL) are popular and thought of as two concrete realization methods of the AI. The objective of ML and DL implementation is that as computers perform labor-intensive repetitive tasks, they simultaneously "learn" from performing those tasks. ML, founded on pattern recognition, has already undergone substantial development over the past twenty years [9]. Generally speaking, ML is a general

topic with a broad scope, and DL can be thought of as one specific branch of it, which has significant representation abilities for learning complex patterns by large amounts of parameters in the deep neural networks. These two approaches have three major categories: (1) unsupervised learning, (2) supervised learning and (3) reinforcement learning based on the data characteristics, where unsupervised learning aims to learn the distribution of data which is label-free; supervised learning uses well-labeled data based on domain knowledge to analyze the input data and produce an inferred mapping function between the data and the label; and reinforcement learning adopts the action-reward data pairs to train the AI agents to take actions in certain environments for maximizing rewards in some tasks [83]. Therefore, both ML and DL fall within the scope of empirical study, and data is the most essential component. In SHM, as the data media, the structural response records and images obtained during post-disaster reconnaissance efforts or daily life play an increasing role under such background, and boost the development of the new research directions using data-driven approaches. The expectation is that eventually computers will be able to realize autonomous recognition of the structural attributes, e.g., damage in daily life, under service conditions, or after an extreme event, like a large earthquake or extreme wind, and then to use this information to assist human to response to the structural health conditions in real time.

Vision-based SHM

Nowadays, structural damage recognition using images is one of the important topics in vision-based SHM and structural reconnaissance, which heavily relies on human visual inspection and experience. Several conventional studies [31, 125, 135, 138] addressed issues of relatively tedious manual efforts and non-autonomous by cataloging key vision features based on human knowledge with respect to damage patterns, e.g., edges of the crack and texture of the concrete spalling area, and then developed algorithms and criteria for determining the damage state based on these features. It is noted that features extracted based on human knowledge are known as *handcrafted features* and the procedure of finding these features is referred to as *feature engineering*. Using handcrafted features for decision making aligns well with the concept and procedure of ML. In other words, it can be considered as using ML algorithms to learn hidden relationships between the extracted features from data and the damage patterns. Handcrafted features are concise, but they may be limited according to the perspective of the image recognition. In some cases, useful damage features are abstract and exist in high-dimensional spaces. Humans are limited by having only a sense of low-dimensional features, e.g., locations, colors, or edges. Due to the complexity and diverse representations of the image data, handcrafted features may not align well with the general cases and usually work for fewer limited scenarios. To address these issues, DL is being considered as a solution to these limitations. Unlike conventional pipelines of feature engineering and ML, the DL approaches uses the concept of representation learning with its abundant number of parameters being proposed to replace the manual feature engineering. In the past decade, great strides have been made by the DL research community, achieving state-of-the-art results in many vision-recognition tasks compared with traditional approaches [38]. Thus, following

the trends of AI and the corresponding improvements of DL in vision-based approaches, it is timely to implement the state-of-art DL technologies in civil engineering applications and evaluate their potential benefits.

Until now, vision-based SHM applications have not fully benefited from the data-driven DL technologies, even with interest in this topic is ever increasing. Its application to structural engineering has been mainly hamstrung due to several factors: (1) the lack of a general automated detection principles or frameworks based on domain knowledge; (2) the lack of benchmark datasets with well-labeled large amounts of data; and (3) the lack of interpretation of the trained DL models.

Past studies were conducted within a limited scope and lack of a systematic approach by simply adopting existing convolutional neural network (CNN) to perform damage state detection without paying attention to procedures of post-disaster reconnaissance or field investigation. Instead of superficially borrowing technologies from computer science, more insights of the engineering domain knowledge should be brought into building a general and systematic vision detection framework, which is reflected by: (1) defining the objectives, i.e., key detection tasks critical for post-event considering inter-task relationships, (2) defining the outputs, i.e., label categories for dataset creation, and (3) developing technical solutions for the framework, i.e., implementation of classification, localization and segmentation algorithms. Therefore, detection tasks can be systematically performed in a consistent manner to engineering design and evaluation. In addition, structural images typically contain large amount of information, defined as multiple structural attributes by domain expertise, e.g., structural component type, damage pattern, or damage severity. Each structural attribute may have a set of labels to describe it. For example, the attribute of component type has labels such as "beam," "column," etc., and the attribute of damage pattern is described by "minor damage," "heavy damage," etc. Thus, detection tasks here are related to recognizing the label for each attribute.

In the computer vision (CV) domain, there exists several large-scale open-source datasets, e.g., ImageNet [24], MNIST [60], and CIFAR-10 [57], with about 15 million, 70,000, and 70,000 images, respectively, and the training of DL models benefit from these large-scale image datasets. However, in structural engineering, especially in vision-based SHM, there is no such large-scale high-variety open-source image dataset until now. It is partially due to the cost and time in collecting and labelling data requiring domain-specific knowledge, e.g., to understand structural component types, damage patterns, etc. Most past studies [14, 26, 68, 130, 131, 141] related to the image-based SHM using DL conducted validation experiments based on their own custom-labeled small-scale datasets and self-defined tasks. As a result, the numerical results reported in previous works are incomparable. Moreover, unlike the above-mentioned CV benchmark datasets, image datasets used in SHM usually have highly imbalanced classes. Such imbalance originates from the nature of SHM: structural damages due to either natural deterioration or extreme events such as earthquakes are less frequent cases due to design code objectives and high levels of conservatism for safety purposes. More recent studies [14, 26, 68, 130, 131] tend to avoid this issue of data imbalance by constructing relatively balanced datasets. Moreover, unlike general CV benchmark datasets

containing common and natural objects, the labeled images in vision-based SHM are much more different. Two major differences are concluded: (1) complex visual patterns, and (2) abstract semantic meaning of the labels to describe the structural attributes, e.g., damage occurrence, damage type, structural component type, etc. As for (1), damage in the form of cracks and concrete spalling do not have fixed shapes and widely vary. For (2), labels in the task to identify damage types, e.g., shear versus flexural damage defined in [82] must engage domain knowledge in structural engineering. For example, flexural damage is attributed to cracks developed horizontally or vertically in the structural component due to the induced bending moments or attributed to cracks or spalling concentrating at the ends of the component, e.g., column, due to plastic hinging also induced by bending moments. Both visual patterns are significantly different, but they have the same semantic meaning leading to flexural damage. Thus, it brings difficulties in training DL models if only a limited amount of data is available.

Transfer learning (TL), as a new ML technique to mitigate the shortage of labeled data, attracts vast attention in both research and industry applications. It applies the knowledge from source domains to target domains which might be related but different [93], making several pre-trained DL models, e.g., VGGNet (Visual Geometry Group) [113], InceptionNet [122], and Residual Network (ResNet) [42], more useful toward other datasets. The major advantage of TL is that it can relax the requirement that training a deep CNN needs a large amount of data, through tuning part of the parameters from pre-trained model in the source domain with few labeled data in the target domain, which might lead to a good performance for the target dataset. Many experiments were conducted to demonstrate the efficiency and promising results of the application of the TL [7, 92, 93]. In TL, there are usually two common strategies: (1) feature extractor and (2) fine-tuning. For feature extractor, all parameters in the net before the fully connected (fc-) layers [38] are frozen, tensors from the last layer before the fc-layers are extracted and flattened as features, which are trained by classifiers such as multi-layer perceptron (MLP) or support vector machine (SVM). For fine-tuning, only some parts of the parameters in the network are frozen and the remaining parameters are retrained with gradient descent and backpropagation (these methods are introduced in Chapter 2). Based on the size of the data, problem complexity and detection expectation, the above two strategies can be applied in different situations as discussed in Chapter 5. However, to the best of the author's knowledge, there are no studies investing the effeteness of using TL in vision-based SHM.

Since decision-making based on SHM has major socioeconomic impacts, clear understanding and explanation of the deep CNN model will yield a more convincing AI for practical engineering applications. The difficulties in explaining deep CNN originate from its nature of the recognition ability. While learning complex patterns from input data, deep CNN determines a set of optimal parameters in the heavily parameterized network, where the learning procedure is numerically performed by updating forward activated information signals and backward gradients to minimize the loss of the objective function. This intensively numerical approach makes it hard to directly perceive the discrimination principle of the network and understand how it really works. Thus, the learning procedure of DL is known as

a "black box." Such shortcoming is very critical to the application of DL in SHM. Although an uninterpretable DL model may achieve an acceptable performance on offline custom validation datasets, without a good understanding of its discrimination principle, it may result in wrong predictions on unseen real-world data. This can lead to serious consequences, e.g., failure to identify shear damage in columns may lead to sudden story collapse during post-disaster evaluation. Similar situations are encountered in the medical imaging [98], self-driving cars [52], etc., where explanation of the DL is equally important to the obtained accurate recognition results.

In summary, in order to address the previously identified three major drawbacks, an automated detection framework along with multiple benchmark detection tasks is expected to be defined and validated for relevant studies and applications. According to the framework and task definition, a relatively large-scale high-variety open-source benchmark dataset is expected to be developed. In addition, more insights should be placed to the interpretation and understanding of the trained deep CNN models, which help relieve the black-box concerns and also give more confidence to the human experts for consideration of the DL models in practical engineering applications.

Advanced explorations in vision-based SHM

Specifically, among the above-mentioned three major issues in use of DL for SHM, data deficiency is usually thought of as the most troublesome one in vision-based SHM. From the past experience in the CV domain, besides the TL introduced above, data augmentation (DA) is one of the other common strategies to mitigate data deficiency issues in current DL applications. In DA, the number of training samples is increased via a series of affine transformations or other preprocessing operations, e.g., translation, flip, scale, whitening, and adding noise. However, both TL and DA have significant curtailments: the former requires fine-tuning a deep pre-trained network with a large amount of parameters, which not only requires a pre-trained open-source model but is also computationally expensive; the latter can only generate highly-correlated data, which does not increase the data variety and sometimes might lead to worse results. Thus, in order to pursue more variety in the data, generative model-based methods are more promising and can be seen as an alternative to TL and DA, where Generative Adversarial Network (GAN) [39] is the state-of-the-art approach.

GAN, first introduced in [39], works as a generative model that can be utilized to generate new data from an approximated data distribution. Different from other generative models (e.g., Linear and Quadratic Discriminant Analysis, Gaussian mixture model, and Restricted Boltzmann machine [9, 66]), GAN holds two different networks playing against each other in a zero-sum game while in operation. One network, the generator, produces synthetic data from some learned distributions by feeding in some latent random variables, and the other network, the discriminator, judges the true identity of the data in a batch flow to determine whether it is real or synthetic. Two types of loss are computed for the generator and the discriminator and optimized iteratively until the two networks converge to optima. In this zero-sum game, the Minimax solution [27] is the Nash equilibrium [37], which indicates that

the discriminator can no long distinguish whether the data is from a real dataset or it is synthetic.

Based on an extensive literature search, to the best of the author's knowledge, there are no GAN-related studies conducted for vision-based SHM. Therefore, as a pioneering study, it is valuable to explore these three questions: (1) whether GAN can be applied in vision-based SHM to generate reasonable synthetic structural images, (2) how to evaluate the goodness of these generated images, and (3) whether using these generated synthetic data can help enhance the DL model performance in certain detection tasks to alleviate labeled data deficiency. Moreover, besides investigating using ordinary GAN pipeline in classification performance enhancement, training GAN in a semi-supervised learning [102, 118] manner is also worth studying.

Vibration-based SHM

Other than using image data, using vibration signals for damage detection is another important aspect in the study of SHM, namely vibration-based SHM, which receives considerable attention. The fundamentals of this approach of SHM can be addressed in the context of a statistical pattern recognition paradigm [30] based on signals collected from sensors to determine location, type, and extent of the damage to make decisions (e.g., repair strategy) about the health status of the structures. This paradigm includes four steps: (1) operational evaluation, (2) structural data collection, (3) damage-sensitive feature extraction, and (4) statistical model development for classification. Within the research of statistical pattern recognition, damage feature extraction (feature engineering) has major work to be conducted, and many studies focus on how to find, extract, and interpret the damage feature. Since the turn of this century, time series (TS) modeling of vibration signals using a family of auto-regressive (AR) models was found to be effective in damage detection and has been used to capture damage features in structures [64, 72, 80, 87, 88, 90, 115]. However, there are some drawbacks limiting the use of AR series modeling in practice.

The most notable limitation of the AR series is the requirement of stationary input, which is usually hard to achieve in real SHM applications, where the TS data (i.e., vibration signals) collected from sensors are usually non-stationary. Thus, elaborate data preprocessing (e.g., segmenting, de-trending, and de-nosing) and stationarity check are inevitable before modeling, but these methods lack systematic pipeline, and may not guarantee stationarity.

Previous studies [87, 88, 90] designed functions for efficient damage features which consist of coefficients of the AR models. However, the definition of such functions is based on intuitive feature engineering. For general structures, these damage features may not effectively work and a more general approach is needed. As a result, ML is thought to be one possible solution that can resolve the limitation of the AR series.

Data play an important role in developing a data-driven SHM, but the data sources in previous studies are limited, which are usually obtained from reduced-scale experiments [80] or numerical simulations [23] and test experiments with component replacement or section decrease to mimic damage [87, 90]. However, during natural hazards, e.g., earthquakes,

the structural damage may differ from that idealized in these artificial cases. Therefore, past applications are limited, and more realistic damage patterns of full-scale structures due to real or simulated hazards, e.g., using shaking table tests, are needed for data-driven vibration-based SHM methods.

In summary, to address the above-mentioned three issues, it is expected to develop an end-to-end framework, where the inputs are simply TS signals and the outputs are predicted structural damage states or patterns with probabilities. Moreover, in the developing stage, the used model and algorithm in this framework should be validated by experimental data, e.g., using full-scale shaking table tests, which simulate the real structural performance under realistic earthquake loading.

1.2 Related Work

Vision-based SHM

Both ML and DL technologies have been developed rapidly in recent decades, especially in the applications of DL in the CV domain [38]. CNN has been at the heart of recent advances in DL. Compared with traditional CV and ML approaches, CNN no longer requires handcrafted low-level features or feature engineering; millions of parameters inside a typical network are capable of learning countless mid- to high-level image representations with input data obtained from a pixel matrix (tensor) data. Another unique characteristic of deep CNN is its depth of architecture. Many well-designed CNN architectures, e.g., VGGNet, InceptionNet, and ResNet, have demonstrated great performance improvement with a substantial increase of the network depth. That said, issues like degradation and bias-variance tradeoff for very deep networks should be given more attention [42]. Although CNN has been used since the 1990s to solve handwritten digit recognition tasks [60, 61], its recent applications on a much wider scale can be attributed to advances in computing technology, such as the high-performance Graphic Processing Unit (GPU).

The ImageNet Large Scale Visual Recognition Challenge [24], known as the ImageNet Challenge, has played a major role in advertising the CNN applicability. The ImageNet Challenge[1], which has been held for several years, is aimed at evaluating algorithms for object detection and image classification. Currently, more than 15 million images have been collected from a variety of domains and labeled, with over 1000+ classes. In the ImageNet Challenge of 2014, VGGNet, the model developed by VGG at the University of Oxford [113], achieved the first place in localization tasks and second place in classification tasks; the VGG Model not only performed well for ImageNet dataset, but also had good generalization properties applicable to other datasets. In the same event, InceptionNet, also known as GoogleNet [122], adopted a novel inception module in the network, and achieved the first place in both classification and object detection tasks. In the ImageNet Challenge of 2015, ResNet, a new CNN architecture developed by a Microsoft research team [42], achieved the first place in all

[1] `http://www.image-net.org/challenges/LSVRC/`.

main competition tracks, indicating great improvement of recognition accuracy compared to previous models and methods via its "shortcut connection" design. Boosted by the ImageNet Challenge, a continuously increasing number of deep CNN architectures are currently under development to achieve better performance. In the above-mentioned studies using images, several basic tasks are commonly performed, namely image classification, object localization, object detection, and semantic segmentation. If extended to vision/image-based SHM, the classification of the damage states and localization of the damage patterns can be thought of as straightforward applications.

There has been an increasing trend of applying CNN technology in post-disaster reconnaissance and SHM. Soukup and Huber-Mörk [117] applied CNN to detect railway defects. Yeum et al. [137] applied VGG-F (F for "fast") CNN architecture to collapse classification based on large-scale images collected from reconnaissance efforts. Cha et al. [14, 15] used a deep CNN to detect concrete cracks without calculating the defect features and proposed a region-based algorithm for detecting multiple damage types. Zhang et al. [141] designed a new CNN architecture, namely the CrackNet, for pavement crack detection at the pixel level. Vetrivel et al. [127] combined deep CNN and 3D point cloud technologies to detect façade and roof damage of buildings. Xue & Li [132] proposed a region-based full CNN to more efficiently classify and localize tunnel lining defects in geotechnical engineering. Most of these studies have mainly focused on the binary problem of whether the structure is damaged or not and analyzed the images at the pixel level. In contrast, images collected by engineers in more general reconnaissance efforts and SHM problems are usually related to the object level or even represent the whole structure; this makes utilizing current CNN technology more difficult to use without first conducting elaborate image preprocessing. A few researchers have tackled this problem, including Yeum et al. [137], who started from the object or the building levels, instead of the pixel level, based on a variety of reconnaissance images. Subsequently, they studied the collapse classification problem and achieved promising results. But in reconnaissance decision making, conducting more complex tasks is essential and requires further studies to determine the damage types and level of damage.

Most studies have conducted experiments based on a small-scale or low-variety image set. Zhang et al. [141] used 2,000 3D asphalt surface images for pavement crack detection, Liang [68] collected 1,164 bridge-related images to conduct post-disaster inspection of reinforced concrete bridge systems. However, to increase the data, many past studies focusing on pixel level images adopted cropping methods to crop raw high-resolution images into many sub-images. Cha et al. [14] collected 332 raw images taken from buildings with 277 images with 4,928×3,264 pixels for training and 55 test images with 5,888×3,584 pixels for validation, and all training images were further cropped to 40,000 sub-images with 256×256 pixels. Xu et al. [130] collected only 12 3,264×4,928 pixels images from steel box girder of a bridge taken by a consumer-grade camera, then reshaped and cropped to 240,448 sub-images with 24×24 pixels. Similarly, Xu et al. [131] gathered 350 raw images and augmented the dataset to 67,200 64×64 pixels sub-images. Dorafshan et al. [26] obtained 18,000 256×256 pixels sub-images via cropping 100 2,592×4,608 pixels raw images, which are for concrete panels to simulate a reinforced concrete bridge. Although cropping increases the number of data,

it brings drawbacks in training DL models. Limited raw images only contain few cases, similar scenarios, and specified type of structural components. After cropping, the high similarity in texture, lighting condition, etc., between training and test datasets can lead to good experimental results but lack generalization to more unseen scenarios. Thus, any high accuracy obtained on such test dataset may not be reliable and not truly reflect the generalization of the DL model where the model just memorizes the similar data. Moreover, using cropping does not introduce new information with inefficient use of computational resources where the learning of the DL models can directly utilize the full-size images instead of the sub-parts. From the perspective of the dataset scale, the amount of data after cropping in the previous studies is still far smaller than that of ImageNet, MNIST, or CIFAR-10. For structural engineering, there exist online-accessible relatively large image collections like DesignSafe[2], or self-taken and collected datasets. However, most of these datasets lack data variety and are without well-defined labels, requiring additional efforts. As a result, increasing dataset scale along with its variety is necessary. Besides non-uniform detection tasks, the datasets used in previous studies are inconsistent. Lack of shareable and benchmark dataset between researchers sets an impediment to compare results from different research groups for validation and further improvement.

A large-scale open-source well-labeled benchmark dataset of images with different attributes of structures and their components is necessary. To the best of the author's knowledge, such dataset is non-existent for general vision-based SHM. However, there exist some datasets mainly for concrete crack detection, namely CrackForest [108], Tomorrow Road Infrastructure Monitoring and Management (TRIMM) [3] and SDNET2018 [77]. CrakForest and TRIMM collected images from pavement, where vision patterns of cracks are similar to those from structural members but the cause of crack formation is different due to different loading and boundary conditions. In SDNET2018 dataset, other than pavement images, it also contains images of walls and bridge decks. However, the number of raw images is still limited to 54 bridge decks and 72 walls. In general, these datasets are for pixel level crack detection and only contain limited concrete component surface areas. Thus, in decision-making procedure of SHM, more information related to component type, damage severity, etc., are still needed, which can only be extracted from images of larger structural areas.

Visual interpretation methods in DL

Recent research [17, 105, 112, 119, 140, 146] in the CV domain attempt to explain and understand deep CNN through visualization by plotting the *saliency map*, presented as heat or grey-scale depth map, where the colors and shades represent the activeness (importance) of pixels in the image towards the model's final decision. Several studies adopt either gradient-based approaches via manipulating the backward gradient from output score to the input image or feature reconstruction approaches through weighting convolution (Conv) features to identify the most discriminative regions and then up-sampling back to the image pixel space.

[2] `https://www.designsafe-ci.org`.

In recent years, the first efforts were made by Zeiler and Fergus [140] to develop a deconvolution (Deconv) approach using a multi-layer Deconvolution Network (DeconvNet) to project feature activations back to the image pixel space. The results, i.e., saliency maps, are reconstructed from restoring the gradient through a backward pass from the feature maps where only positive gradients are retained. Saliency maps generated by this approach highlight certain areas of the input image which strongly activate the neurons and excite the feature maps at any Conv layers in the network. Instead of only visualizing neurons in the Conv layers and using features learned from multiple images by DeconvNet, Simonyan et al. [112] proposed the image-specific class saliency visualization, which uses the derivative of a class score with respect to one single image. It can be applied to any layer in the network, and most commonly performed in the fc-layer. The derivative herein is a gradient matrix with the same size as the input image and can be easily computed by a single backpropagation (BP) pass, where the gradients are zero if the activation is zero in the forward pass. It is noted that the magnitudes of the entries in the gradient matrix represent the activation condition of each pixel in an image towards the class score, and the backward signal shares the same activation condition of the activation function, e.g., ReLU [89]. Based on the above methods, Springenberg et al. [119] proposed the guided-backpropagation (GBP), which guides the backward signal from the deep layers (last several layers) to the input image by zeroing the entries of the signal having either negative values of the gradient in the backward pass or zero activation in the forward pass. The results produced by GBP show clearer visualizations than the above two methods. However, both DeconvNet and GBP are not class-discriminative.

From the perspective of feature reconstruction, Zhou et al. [146] developed a new approach to highlight the class-discriminative regions, namely class activation map (CAM). Global average pooling (GAP) is applied on the Conv feature maps and the output of the GAP is followed by a new single fc-layer (whose outputs are class scores) replacing the original fc-layers. Through weighted sum of the feature maps and the fc-layer weights for one specific class, CAM generates a class discriminative saliency map. After up-sampling to the image size, the saliency map can identify the most discriminative area of the specific class in a heat map format. However, CAM altering the raw pre-trained model may lose some information in the original fc-layers, which leads to recognition performance deterioration in the modified network. Moreover, to determine the updated weights in the added single fc-layer requires running forward propagation once again, which is computationally more expensive than other methods. To address these drawbacks, Selvaraju et al. [105] extended CAM into a more generic approach, namely Gradient-weighted CAM (Grad-CAM). Instead of modifying the model, Grad-CAM computes the weights of the feature maps by gradients of the class scores with respect to the feature maps in the particular layer of interest. Grad-CAM can produce better interpretable saliency maps with clearer boundaries and more precise object locations than CAM. Furthermore, combining both Grad-CAM and GBP, a more fine-grained high-resolution visualization can be generated, which is known as Guided Grad-CAM. Compared to GBP, CAM, Grad-CAM and Guided Grad-CAM are class-discriminative. Currently, Grad-CAM is the most popular and widely accepted approach in DL model interpretation [67]. There are similar limitations of Grad-CAM to those of CAM, e.g., the most active area cannot

cover the entire target and Grad-CAM performs less satisfactorily if there exist multiple objects of the same class. To overcome these limitations and further enhance the interpretable performance, Chattopadhay et al. [17] modified the formulation of weights computation and formed Grad-CAM++. Their results indicate that Grad-CAM++ algorithm has a more complete coverage towards single or multiple targets. Visual interpretation methods other than gradient-based or feature reconstruction methods include Local Interpretable Model-Agnostic Explanation (LIME) [99], DeepLift [110], etc. More information about these methods can be found in the literature review in [17].

The methods discussed above are dedicated to generating good visualizations with reasonable explanations to a well-trained DL discriminative classifier, where the highlighted pixels or the active regions in the saliency map focus on the object consistent to its class label, which is analogous to human judgement. Therefore, these interpretation methods provide an opportunity to visualize and understand the model, which can partially resolve the "black box" issue. This is expected to inspire and expand the practical engineering applications, e.g., vision-based SHM, through better exploration and understanding of the trained DL models.

In the vision-based SHM, there are no relevant studies specifically focusing on visually understanding and explaining the working principle of the DL models in damage recognition. There exist only very limited early applications by simply adopting visual interpretation method to validate the trained deep CNN without comprehensive explorations. Liu & Zhang [69] used GBP to visualize the feature related patterns identified by a trained CNN for different damage levels, but the qualities of the obtained saliency maps were not satisfactory nor class-discriminative. The active areas of the images using these visual interpretation methods are ambiguous and unclear, and only small portions of the prediction results are reasonably presented and explained. Therefore, a more comprehensive investigation in this direction should be pursued.

GAN in vision-based problems

Other than the ordinary GAN proposed by Goodfellow et al. [39], the rapid growth of GAN studies has produced many types of GANs, which modify the ordinary GAN with different network architectures, loss functions, training strategies, etc. In order to improve the performance of the original GAN, Conditional GAN [81] was proposed by conditioning the models on additional information (e.g., class labels). Radford et al. [97] proposed a class of CNNs called Deep Convolutional GAN (DCGAN) which shares the same architectural constraints and have been proven to be powerful for unsupervised learning. The architectural constraints can contribute to the stable training process across a wide range of datasets. One major shortcoming of DCGAN is mode collapse, where the generator will produce synthetic images with extremely low variety. To address this problem and keep a balance in training the generator and the discriminator, Wasserstein GAN (WGAN) is presented by introducing Wasserstein distance to the loss function [5]. Until now, there already exist over 300 types of GANs.

GANs have been proven to be powerful in many applications. Currently, facial image generation is one of GAN benchmark tests and new direction of facial recognition attracting much attention, e.g., Radford et al. [97], Legid et al. [62] and Berthelot et al. [8] are conducting many experiments with promising outcomes. Moreover, Jin et al. [50] extended previous work towards anime demonstrating the stable performance of GAN to create high-quality anime character faces. Ma & Ma [73] proposed GAN based on Artistic Textual Visualization (GAN-ATV) to convert poems into paintings. Kitchen & Seah [54] introduced GAN to the medical filed, and they applied DCGAN to generate synthetic lesion images similar to real ones. To create Depth Structure Preserving (DSP) scene, Zhang et al. [143] designed DSP-GAN which demonstrated good performance in real-world datasets. Recently, researchers are using GANs to augment datasets by generating synthetic images in order to improve the model performance. Antoniou et al. [4] proposed Data Augmentation GAN (DAGAN) and compared the performance of DAGAN trained classifier with other basic ones based on Omniglot, EMNIST and VGG-Face datasets, showing promising enhancements. Mariani et al. [78] proposed Balancing GANs (BAGANs) as an augmentation tool to restore balance in imbalanced datasets such that it can learn useful features from the majority classes and uses these to generate images for the minority classes. In the medical imaging community, Frid-Adar et al. [32] applied DCGAN to generate synthetic medical images for three lesion classes (Cysts, Metastases, and Hemangiomas) and then augmented the real image dataset with the generated images, which achieved 7% improvement over the classical augmentation methods. Similarly, for cardiovascular abnormality detection, Madani et al. [75] first down-sampled Chest X-Ray images to 128×128 pixels and then used a variant of GAN to generate synthetic images to enrich the raw dataset. With the mixed images, the DL classifier reached a roughly 2% improvement over the non-augmented baseline and 1% over the traditional DA. Furthermore, Madani et al. [76] formatted the GAN in a semi-supervised learning manner and conducted comparative experiments to investigate the effectiveness of the GAN under different levels of labeled data. Their results indicated that such GAN worked more efficiently under low-data regime.

Based on an extensive literature search, to the best of the author's knowledge, there are no GAN-related studies conducted for vision-based SHM. Therefore, as a pioneering study, it is valuable to conduct more research in this direction.

Vibration-based SHM

Different from the image data used in vision-based SHM, the data source in the vibration-based SHM is the TS signal, where data points are discrete and follow certain sequence with hidden relationships. TS modeling is known as conventional tool and approach to fit and predict the TS data, which aims to derive the hidden relationships and then to make informed decision making. Since 2000, TS modeling of vibration signals was found to be effective in damage detection and have been used to capture damage in structure. Sohn et al. [115] proposed to apply TS analysis in structural damage identification and their approach led to promising results in identifying damage in the hull of a high-speed patrol boat. According

to Mei et al. [80], past studies using AR series models are grouped into: (1) residual-based and (2) coefficient-based, where the former identifies damage through measurement of the residuals computed from the difference between actual data and the fitted data, and the latter uses the coefficients of fitted model as the damage feature.

The residual-based approach is founded on the fact that a good fitted model trained by undamaged data is not a good fit for damaged data, and residuals increase with the increase of the damage. Sohn & Farrar [114] constructed a two-stage prediction model combining AR and AR with exogenous (ARX) models for fitting acceleration signals. They defined a damage-sensitive feature related to the standard deviation of the residual error, which is computed as the difference between actual data and the fitted results by the AR-ARX model. Lynch et al. [72] designed a wireless unit including implementation of the Fast Fourier Transform (FFT) and TS modeling. In this unit, ARX input model is used to fit the absolute acceleration and also using a damage feature related to the ratio of the standard deviations of the residuals of the ARX model obtained from the accelerations of the undamaged and damaged structure. The damage state is identified if this ratio exceeds a threshold value. Lei et al. [64] proposed a detection algorithm based on Sohn & Farrar [114], which demonstrated good accuracy for medium and severe damage detection and localization, but lower accuracy for minor damage. Subsequently, Nair et al. [88] improved this algorithm using a normalized relative inter-story acceleration instead of a single floor acceleration, making the algorithm more robust and capable of detecting minor damage patterns. Other studies [28, 33, 79] modified the processing procedure and format of the residuals for simulated or experimental data.

The coefficient-based modeling is shown to be effective in many TS studies. Nair et al. [87] proposed a new algorithm using auto-regressive moving average (ARMA) model for the vibration signals with a damage feature expressed as a function of the first three AR coefficients, and localization indices were introduced based on the AR coefficients. The algorithm detected the existence of all damage patterns including minor ones. In addition, Noh et al. [90] selected the first three regressive coefficients as the feature vector, then used a Gaussian mixture model and Mahalanobis distance to successfully detect, quantify, and localize the damage. Xing & Mita [129] proposed a substructure approach that combined ARMA with exogenous model (ARMAX) to identify localized damage in any story of a shear building with limited number of sensors. Recently, Mei et al. [80] combined both coefficient-based and residual-based approaches with ARMAX model and used normalized Kolmogorov-Smirnov statistical distance between two sets of ARMAX model residuals obtained from the input-output process for the undamaged and damaged states. They validated their approach on a small-scale five-story frame using aluminum floor slabs and bronze columns. Besides pure TS methods, Lakshmi et al. [59] combined singular spectrum analysis (SSA) with ARMAX model to detect the small damage, e.g., minor cracks in concrete structures.

There exists one special model in the AR family, namely auto-regressive integrated moving average (ARIMA) model. Unlike conventional AR models, ARIMA is developed by performing integration (differencing) of the TS, which helps to achieve stationarity effectively [111]. However, there are very few cases in the literature that used ARIMA in SHM.

Omenzetter & Brownjohn [91] formulated a vector seasonal ARIMA model whose coefficients vary with time and were identified using adaptive Kalman filter. Through analysis of signals recorded during and after construction of an instrumented bridge, they observed changes in the model coefficients and concluded that these changes were related to damage under normal operational and environmental conditions. Recently, Yang & Bai [133] compared results between SSA and ARIMA in the problem of forecasting short-term and long-term structural strain variations.

Nowadays, the implementation of ML in vibration-based SHM pattern recognition is attracting more attentions in the structural engineering community, especially using supervised learning by data and label pair, which learns hidden relationships between some features extracted from the data and the corresponding label describing the damaged state of the structure. An early exploration in this direction was performed by de Lautour & Omenzetter [23]. They numerically simulated a three-story shear building and successfully used coefficients fitted from floor accelerations by the AR model as input features of a neural network (NN) to detect and locate damage of the building. However, there are no recent studies focusing on promoting the concept of building an end-to-end SHM framework by TS modeling in conjunction with ML classifiers, which is worthy of further study.

According to the data type, the previous studies can be mainly categorized as using simulation data, reduced-scale experimental data, and full-scale experimental data. A few examples are reviewed here. Mei et al. [80] used data from a numerically simulated structure and a laboratory test of a reduced-scale five-story frame, and damage patterns were designed by replacing some components. The data used in [87, 90] were from full-scale ASCE[3] benchmark steel frame tests for damage patterns, artificially designed by decreasing cross-sectional areas to simulate damage. Other than building structures, [29, 91] used data from full-scale instrumented bridges under normal operational and environmental conditions with small damage levels. However, most previous studies relied on the numerical simulation results and few studies considered to conduct full-scale validation experiments. Furthermore, almost all research adopted the idea of mimicking damage patterns by replacing component with lower stiffness or decreasing sections at designated places. Actually, these patterns are ideal and may not reflect the true situations after several random loading events, e.g., earthquakes. As a result, using such data may exaggerate the performance of the validated algorithms. More efforts are expected to be made in collecting data representing realistic damage patterns.

1.3 Objectives & Contributions

The background and related work discussed above reveal the great potential of introducing the state-of-the-art ML and DL technologies into the SHM domain. This dissertation investigates the fundamentals and novel applications of ML and DL in data-driven SHM. Based on the data type, two major directions in data-driven SHM are explored, namely vision-based SHM

[3]American Society of Civil Engineers.

and vibration-based SHM. Moreover, since using ML and DL in vision-based is more active than in vibration-based SHM, more efforts are devoted to the vision-based SHM direction. Therefore, the vision-based study is further split into general and advanced topics. The detailed objectives and contributions for these parts are presented in the following subsections.

Contributions in general topics of vision-based SHM

In the general scope of vision-based SHM, major drawbacks and objectives are summarized as follows:

- Lack of an automated detection framework with basic/benchmark detection tasks;
- Lack of a publicly open-source and relatively large-scale high-variety labeled structural image dataset; and
- Lack of understanding and interpretation of the trained DL models for structural recognition problems.

The contributions of this dissertation in order to address these general issues and explore DL performance in real application scenarios are listed below:

- **Define benchmark vision tasks according to key structural attributes and propose corresponding automated detection framework in a hierarchy format.** Based on past experiences from reconnaissance efforts of post-earthquakes and other hazards, a hierarchy tree framework namely PEER Hub ImageNet (ϕ-Net or PHI-Net) was established. Through a tree-branch mechanism, each structural image can be clustered into several sub-attributes, e.g., scene level, damage state, and component type, by detection tasks, which work as filtering operations to decrease the complexity of the problem and improve the performance of the automated algorithms. As a preliminary study, within the ϕ-Net framework, eight basic tasks are selected for recognizing key attributes, defined as benchmark and tested in this dissertation.
- **Collect, process and label images to form a relatively large-scale multi-attribute dataset with high data variety, and then open-source it.** Through crowdsourcing, a large number of images were collected and labeled, and the current version of the ϕ-Net dataset contains 36,413 images with multiple attributes. The dataset is publicly released in `https://apps.peer.berkeley.edu/phi-net`. Compared to previous works, the ϕ-Net dataset is thus the first open-source labeled structural image dataset containing various and complex vision patterns of structural components, scenes, damage states, etc. where the labels of each image can directly contribute to the classification problem. It will serve as a benchmark dataset and foundation for future studies and activities of DL related to vision-based SHM.

- **Conduct and report the reference results in each vision task trained by DL models for benchmarking purpose.** The experiments using the same setting for each task with multiple training strategies, i.e., TL and DA, were conducted to obtain benchmark performance. All training histories with respect to loss and accuracy, and confusion matrices were presented and analyzed. Researchers are encouraged to test newly developed algorithms or models using the ϕ-Net dataset with corresponding tasks and compare their performance with the benchmark results reported herein. In addition, the optimal DL model in each task, namely the Structural ImageNet Model (SIM), was also saved for future studies.

- **Apply visual interpretation approaches to understand and validate the DL model performance in vision-based SHM.** In this part, the study emphasized the importance of understanding the trained deep CNN in the vision-based SHM and investigated the applications of the recent widely-used visual interpretable methods, namely GBP, Grad-CAM, and Guided Grad-CAM, for the basic vision task of SHM. Beyond the general view over the non-expert spalling condition detection task, several key factors in the DL training were investigated, namely learning performance at different epochs, feature maps extracted from different network depths, influence of TL and DA, and different levels of task abstraction.

- **Two applications with respect to ϕ-Net were explored.** The first is an image-based post-disaster damage assessment of the 1999 Chi-Chi earthquake where SIMs trained from the benchmark experiments were directly applied to extract attributes from images collected after the earthquake. An acceptable performance in both accuracy and running time was obtained, which can further provide information for decision-making purposes. The second is the ϕ-Net Challenge held prior to open-sourcing the ϕ-Net dataset. It is the first image-based challenge in structural engineering, which encouraged *transdisciplinary* collaboration.

Contributions in advanced topics of vision-based SHM

Beyond the above-mentioned three general issues, there are several additional topics that can be further explored, as listed below:

– TL and DA are used as conventional strategies to alleviate data deficit. However, in some cases, e.g., low-data regime and limited computational resources, TL is inaccessible and DA may even render worse results than without DA. Thus, GAN-based augmentation method can be studied in vision-based SHM.

– Until now, there are no GAN-related studies in vision-based SHM. Therefore, the first exploration study is needed, e.g., to validate whether GAN can generate realistic structural images with and without damage patterns and validate whether GAN-based

augmentation techniques can enhance the DL model performance in certain tasks, e.g., classification.

– Besides data deficiency, structural image datasets are usually highly imbalanced, where damage-related images are far less than the undamaged ones, which may yield extreme biased models. Such issue is often ignored in the previous studies.

– More advanced network structure and training strategies can be introduced into the GAN pipeline to improve the efficiency and performance, especially for imbalanced datasets.

Motivated by the needs of the above-mentioned specific issues in the vision-based SHM, this study additionally contributed the following:

- **Introduce GAN into vision-based SHM to investigate the performance of GAN in generating realistic structural images.** In this part, the study led to proposing simplified representations of the ϕ-Net as the testbed, designing a specific GAN model suitable for generating synthetic structural images, proposing Leaf-Bootstrapping algorithm to improve the GAN performance for the developed custom dataset, proposing reasonable and quantitative metrics for GAN performance evaluation, and conducting experiments to examine the GAN performance in synthetic structural image generation within the scope of scene level identification and damage state check.

- **Validate the effectiveness of using GAN augmentation to enhance the DL performance under the conditions of low-data regime and limited computational power.** A special case representing these constrictions was designed, different factors were considered as comparisons, namely influence of using TL and DA in both baseline model and GAN model, and different training pipelines utilizing GAN generated synthetic images.

- **Reformulate the GAN pipeline into a semi-supervised learning framework, and further investigate it on imbalanced-class dataset problem on the basis of the above two contributions.** In this part, a Semi-supervised GAN model using balanced batch sampling technique was developed for structural damage assessment. It is named as Balanced Semi-supervised GAN (BSS-GAN) and its performance was evaluated and compared with the baseline DL classifier using conventional training through multiple comparative experiments.

Contributions in vibration-based SHM

According to previous studies in the vibration-based SHM using TS modeling techniques, several limitations are concluded as follows:

- Stationarity is usually difficult to achieve from the collected TS data in real SHM applications, where stationarity is essential for TS modeling.
- Conventional methods through feature engineering to determine the damage features for damage state/pattern are based on intuition, making these features non-comprehensive and the SHM process may lose important information.
- Few studies attempted to combine both TS modeling and ML techniques. Thus, there is a lack of efforts to formulate this integration into an end-to-end framework.
- Data used in previous studies are from idealized cases, e.g., reduced-scale experiments or numerical models with component replacement or section reduction, which are not realistic enough to mimic damage in real situations caused by natural hazards, e.g., earthquakes.

The above needs motivated further contributions of this study as listed in the following:

- **Introduce ARIMA modeling into vibration-based SHM to relax the stationarity requirement.** Several basics of TS modeling and AR series models, especially the ARIMA model, were covered. Model parameter selection criteria accompanied with exploratory data analysis (EDA) was also introduced.
- **Develop an end-to-end framework/algorithm, namely ARIMA-ML, to combine ARIMA modeling techniques and ML approaches for detecting damage patterns.** The framework packaged several modules, namely smoothing-segmentation-normalization-differencing (SSND), auto-stationary check, ARIMA feature extraction, and ML classification. Compared to conventional methods, instead of using intuition and trial and error to determine a suitable function or a subset of the ARIMA coefficients as damage features, all the coefficients are used as damage features, which builds a $p + q$ dimensional vector space, where p and q are respectively the number of coefficients in the AR and moving average (MA) parts of the ARIMA model.
- **Conduct validation experiments for the developed ARIMA-ML using data from full-scale shaking table tests of a three-story steel braced frame.** In the numerical experiments making use of actual data from full-scale shaking table tests, different ML classifiers, e.g., SVM, NN, random forest (RF), and logistic regression (LR), were first applied individually and then an ensemble voting classifier (EVC) was applied and compared with individual classifiers. Finally, feature importance (FI) was analyzed to examine the most important features for damage detection and pattern recognition, illustrating the need for higher order coefficients and validating superiority of the proposed end-to-end framework.

1.4 Organization of the Dissertation

The dissertation is organized as follows:

- Chapter 1 introduces the background, related work, and the objectives of this research.
- Chapters 2 & 3 review the basics of ML and DL, respectively, which provide the technical and theoretical foundation for the subsequent parts of the study.
- Chapter 4 introduces the buildup of a large-scale high-variety open-source Structural ImageNet (ϕ-Net) framework and dataset along with benchmark tasks and several framework variations.
- Chapter 5 investigates the experiment of using TL on a small-scale dataset as an early validation of the effectiveness of DL in vision-based SHM.
- Chapter 6 conducts comprehensive benchmark experiments on the ϕ-Net with reported results, which provide references for related studies.
- Chapter 7 introduces two applications and extensions with respect to the ϕ-Net, namely image-based post-disaster damage assessment using ϕ-Net pre-trained model in 1999 Chi-Chi earthquake and the 2018 ϕ-Net Challenge.
- Chapter 8 reviews the fundamentals of the DL visual interpretation methods, and introduces them into the vision-based SHM domain to partially address the "black box" issues of DL.
- Chapter 9 conducts comprehensive experiments to explore the performance of visual interpretation methods in vision-based SHM.
- Chapter 10 introduces the principle and mathematical formulation of GAN and proposes the LB training algorithm for better GAN performance towards structural images.
- Chapter 11 conducts numerical experiments using GAN to generate synthetic structural images for two scenarios, and also evaluates the quality of the results based on defined metrics.
- Chapter 12 introduces the formulation of the BSS-GAN and conducts comparative experiments to discuss the effectiveness and efficiency of the BSS-GAN over baseline and other GAN-based augmentation methods.
- Chapter 13 reviews the basics of TS including the formulation of the AR family model, and proposes the ARIMA-ML framework.
- Chapter 14 performs EDA on collected data to select several best ARIMA candidate models and conducts numerical experiments through three different recognition tasks.

- Chapter 15 enlists conclusions for several findings in this dissertation and also proposes extensions for future studies.

From the perspective of logic, the dissertation can be thought of as consisting of five parts. Besides the introductory Chapters 1 to 3 (Part 1) and conclusive Chapter 15 (Part 5), Chapters 4 to 9 (Part 2) include the material about using DL in general problems of vision-based SHM, Chapters 10 to 12 (Part 3) include the material about advanced topics, i.e., GAN techniques, in vision-based SHM, and Chapters 13 & 14 (Part 4) include the material about using ML in vibration-based SHM. The dissertation is intended to present the foundation and applications of ML and DL in different research directions as they relate to the important and modern topic of the data-driven SHM.

Chapter 2

Basics of Machine Learning

In the past decades, machine learning (ML) has achieved significant advances in both theory and application. Generally speaking, ML has three major categories: supervised learning, unsupervised learning, and reinforcement learning based on data characteristics. Supervised learning trains a model that can learn and infer the mapping function between the input data and their corresponding labels. Compared to the other two categories, supervised learning is the most active branch in ML research and is widely-used in the current ML applications.

In this dissertation, the study mainly focuses on supervised learning, and some background related to unsupervised learning is also covered in specific topics. This chapter introduces a brief review of the basics of ML with respect to theoretical and mathematical fundamentals. More details can be found in [9, 66].

2.1 Supervised Learning

Basic concepts & notations

In supervised learning, the collections of all possible values for input and output are defined as *input space* and *output space*, respectively, where both spaces can be finite or the entire Euclid space. Each concrete realization of the input is named as *instance*, and it is usually represented by the *feature vector*. The space formed by the feature vectors is called *feature space*, and each dimension in the feature space corresponds to one feature. The input space and the feature space can either share the same space, or just be different. If it is the latter, one instance can be mapped from the input space to the feature space through some mapping or transfer functions.

Based on statistics, the input and output can be thought of as two random variables in the input (feature) space and output space, where input and output variables are denoted as X and Y, respectively. The values taken by the input and output variables are denoted as x and y, respectively. The variable can be either a vector or a scalar. For example, the vector form of the i-th instance of the d-dimensional x is represented as:

$$x_i = ({x_i}^1, {x_i}^2, \ldots, {x_i}^j, \ldots, {x_i}^d)^T \tag{2.1}$$

where ${x_i}^j$ represents the j-th feature of the i-th instance of x.

Random variables X and Y can be of different types, i.e., continuous or discrete, to form different prediction tasks and scenarios. For example, if the output variable Y is discrete, the prediction task is defined as classification, which is a major focus of this dissertation. Based on different variable types, other tasks, e.g., regression, can also be formed.

In supervised learning, a collection of the n pairs (x_i, y_i), known as a *sample* or a *sample point*, is first constructed as the *dataset*, D, and expressed as:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{2.2}$$

To train and evaluate the model, the dataset D is further split into *training set*, *validation set*, and *test set*. The training set is used to train the model parameters to find the optimal mapping function between the input and output pairs used in the training. The validation set is used to find the best model configuration among multiple candidates from varying model parameters. The test set is used to evaluate the model on the unseen data (not used in the training), which reflects the model's true generalization ability. In many cases, e.g., research exploration, the split of dataset can be simplified to only have training and test sets to better exploit limited data.

Hypothesis model

As mentioned above, the key in supervised learning is to make the model learn the mapping function from input to output. In other words, its major objective is to find the best model to capture such relationship, and such model is named *hypothesis model* and denoted as $h(x; D)$. Statistically, the hypothesis model $h(.)$ is also a random variable, where $h(x; D)$ depends on arbitrary input x and dataset D.

Based on specific learning algorithms and application scenarios, the hypothesis model can be represented as either a probability model or a non-probability model. For concrete instance x and y, the probability model is to learn the conditional probability $P(y|x)$ based on the input x, which can also be thought of as observations; and the non-probability model is to learn the decision function $y = f(x)$.

Classification

As mentioned above, classification is one kind of learning tasks and is based on the data type, where the output Y is a discrete variable. Suppose now a dataset D is collected, for each input x_i, which is a d-dimensional data (feature) vector, it is assigned with one discrete class label $c_i \in \{1, 2, \ldots, K\}$ among K possible classes.

In a classification problem, the model is also known as the *classifier* in the literature. With the input, the classifier outputs a K-dimensional vector of posterior probability of the

class label, $P(y|x)$. The predicted class label, c^*, is assigned by the one having the highest probability among all the K classes:

$$c^* = \underset{c \in \{1,\dots,K\}}{\operatorname{argmax}} P(y = c|x) \tag{2.3}$$

There exist many well-known classification algorithms and models [9], and they are discussed in details in the subsequent sections.

Loss function

During the training procedure, to maximize the performance, the classifier and its parameters are expected to be optimized based on the *objective function*. It is usual to conduct minimization on this objective function. Therefore, it is usually referred to as the *cost function* or the *loss function* instead.

The loss function evaluates the degree of the prediction error made by a trained model between the prediction $f(x)$ and the *ground truth*[1] y, denoted as $L(f(x), y)$. There are many types of the loss function that can be utilized for different tasks. For example, in classification, the commonly used loss functions are 0-1 loss, logarithmic loss (usually using the natural log function), and cross-entropy loss. On the other hand, in regression, commonly used ones are mean square error (MSE), mean absolute error (MAE), and root mean square error (RMSE). Some of these metrics related to the discussion in the subsequent studies are introduced as follows:

- 0-1 loss

$$L(f(x), y) = \begin{cases} 1 & y \neq f(x) \\ 0 & y = f(x) \end{cases} \tag{2.4}$$

- Cross-entropy loss

$$L(f(x), y) = H(\hat{y}, y) = -\sum_{\text{all classes}} p(y) \log(q(\hat{y})) \tag{2.5}$$

  where $\hat{y} = f(x)$, $p(y)$ and $q(\hat{y})$ are the probability distributions of the true labels and the predictions, respectively.

- Cross-entropy loss (Binary classification)

$$L(f(x), y) = -y \log(p(\hat{y})) - (1 - y) \log(1 - p(\hat{y})) \tag{2.6}$$

  where the ground truth y is either 0 or 1, and $p(\hat{y})$ is the probability distribution of the predictions.

[1] In ML, ground truth is factual data that have been observed, labeled, or measured, and can be analyzed objectively. Here, it refers to the true label.

Metrics

Starting from the binary classification, several definitions are made first. In statistics, condition *Positive* is the number of positive cases in the data, e.g., corresponding to the state (label) that the building is *damaged*. Similarly, condition *Negative* is the the number of negative cases, e.g., corresponding to the state (label) that the building is *undamaged*. *True* and *False* are two judgements of the model prediction, and yield four possible outcomes: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

A TP means that the model correctly predicts the positive class, e.g., if the column is damaged, and the model also predicts it as damaged, and FP represents the model incorrectly predicts the negative class, e.g., if the column is undamaged (negative), but the model predicts it as damaged (positive). Similarly, a TN means that the model correctly predicts the negative class and a FN means that the model incorrectly predicts the positive class.

Overall accuracy

In the multi-class classification, there are many metrics to evaluate the goodness of the results. The most common one is *overall accuracy* (referred to as accuracy in some publications for brevity), which is defined as the ratio between number of correct predictions to the total number of data used for prediction, Eq. 2.7. Furthermore, in a binary classification, it can be rewritten in a another form using the above defined four outcomes, Eq. 2.8.

$$Overall\ Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ total\ predictions} \tag{2.7}$$

$$= \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

As its name suggests, the overall accuracy represents a general view of the accuracy of the model, but it may not reflect the true situation under a biased dataset, where the influence of the majority class dominates the minority. For example, if 90% images are describing as undamaged state of the building and only 10% images are damage related, a naive classifier that simply predicts all images are in the undamaged state will yield a 90% overall accuracy. The value is indeed promising, but the result is nevertheless dangerous. Because in practice, the main objective of SHM is to identify structural damage accurately, and herein overall accuracy can not correctly satisfy such goal based on a biased dataset. Therefore, some other metrics are needed to complement the results of the overall accuracy in some cases.

Confusion matrix

To investigate the bias issue, confusion matrix (CM) [56] is usually constructed based on the above-mentioned four outcomes, Fig. 2.1. In this dissertation, the rows in the CM represent the ground truth, e.g., the first row represents the number of data whose true labels are positive (e.g., damaged). On the other hand, the columns in the CM represent the predictions,

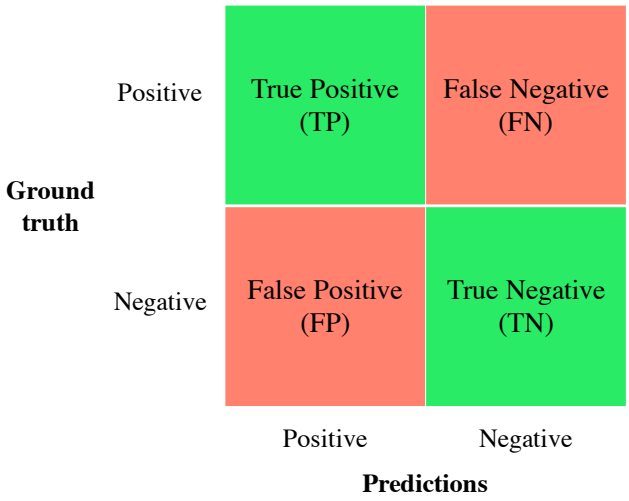


Figure 2.1: Confusion matrix

e.g., the first column represents the number of data predicted to be positive (e.g., damaged) by the ML model or algorithm.

Utilizing the relationships between the above mentioned four outcomes leads to other useful metrics in a binary classification, namely *precision* and *recall*. Precision is calculated in Eq. 2.9, which is the ratio between TP and the sum of TP and FP. It presents the model ability to identify the true positive data among all data predicted as positive. Take an example of a binary classification in a building damage identification. Suppose the computed precision of the model is 0.6, i.e., if the building is predicted as damaged by the model, it is expected to be correct 60% of the times.

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

Recall is also known as *true positive rate* (TPR) or *sensitivity*, Eq. 2.10, which measures the model recognition ability towards the positive cases. Taking the same example above, suppose the computed TPR is 0.6, i.e, the model can identify 60% buildings which are damaged.

$$Recall\ (TPR) = \frac{TP}{TP + FN} \tag{2.10}$$

Both precision and TPR are useful to evaluate the model performance in a biased dataset, where some intuitive preference is given to using the TPR. Due to the nature of the SHM, damage occurring in a structure is less frequent but can be very dangerous and may lead to significant loss (causalities (i.e., life loss), monetary (i.e., economical loss), or downtime (i.e., business loss)). Therefore, in practice, engineers may tolerant possibilities of FP, and try to eliminate FN as much as possible. In other words, a model with a high TPR is thought to be more conservative, and it is more desirable if accompanied with high overall accuracy and precision.

Beyond the above-mentioned metrics, there are other ones, e.g., F1-score, receiver operating characteristic (ROC) curve, and the Area Under the ROC curve (AUC). These metrics can be found in [9, 66] and other statistics related references.

Extending to a multi-class classification (i.e., number of classes is larger than 2), the overall accuracy and CM are still good and reliable metrics. The CM, in particular, by normalizing each row of the CM, the entries along the diagonal of the CM represent the recall (TPR) for each class. The normalized CM provides an easy evaluation of the model recognition ability towards the specific classes. More concrete examples are shown in the computer experiments presented in the later parts of this dissertation.

Over-fitting & under-fitting

The main objective of training a ML model is to adapt it towards new and previously unseen data, and the ability of the model to recognize these unobserved data is called *generalization*. As mentioned above, in several ML research activities and practical applications, the collected dataset is usually divided into training set, validation set, and test set or just training and test sets. Thus, a good generalization is reflected by the fact that the ML model can achieve: (1) good performance on the training set (small training error), and (2) equivalently good performance on the unseen test set (small test error and small gap between both training and test errors). Equivalently, having a small error is the same as having a high accuracy.

There are two common challenges in the ML applications, namely *over-fitting* and *under-fitting*. The over-fitting issue leads to a large gap between training and test errors, i.e., the test error can be much larger than the training error. Moreover, Over-fitting indicates that the model lacks generalization, and it usually over-emphasizes the training data with a small training error and may even memorize them instead of learning the true distribution of the data. On the other hand, under-fitting is attributed to the model being unable to achieve sufficiently small errors on either the training set or obviously the test set. An example of appropriate fitting, over-fitting, and under-fitting using the same datasets is presented in Fig. 2.2 for illustration.

Cross-validation

In ML models (algorithms), besides model parameters to be found during training, there also exist a set of parameters needed to be determined before training based on prior knowledge, which are referred to as *hyper-parameters*. The usage of these hyper-parameters usually directly contributes to the model performance, e.g., some poor choices of the hyper-parameters may cause over- and under-fitting issues.

In order to obtain a set of optimal hyper-parameters to avoid loss of generality, usually *cross-validation*, also known as *k-fold cross-validation*, is applied. In the cross-validation, all data is randomly and equally split into k folds, and for each time, $k-1$ folds with a new combination of folds are used as a training set and the remaining fold is assigned as a test set until each fold has been tested once. Conventionally, there are two adopted patterns

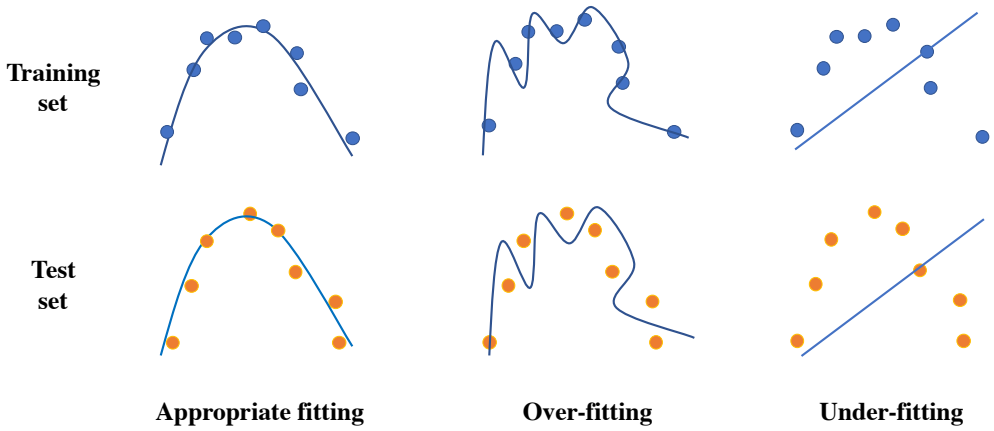
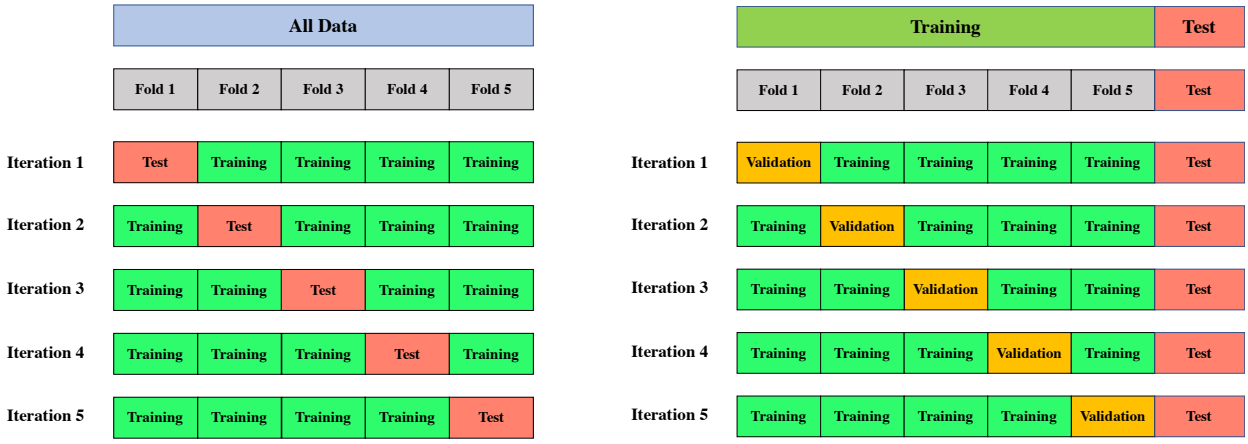


Figure 2.2: Illustration of appropriate fitting, over-fitting, and under-fitting



(a) Pattern 1 (b) Pattern 2

Figure 2.3: Example of a 5-fold cross-validation

in cross-validation under different conditions, namely (1) separate all data to be training and test sets when the collected dataset has not specified the test dataset, and (2) if the collected dataset has already been separated into training and test sets, then one further separates the training set only into training and validation sets. Two examples of 5-fold ($k = 5$) cross-validations for the two patterns 1 & 2 above are illustrated in Figs 2.3a & 2.3b, respectively.

2.2 Logistic Regression

Logistic regression (LR) is one of the basic ML algorithms for classification, which has a direct probabilistic meaning. Starting from the setting of a binary classification, the two

classes are represented by 0 and 1. The model conducts a linear mapping, $f(x) = w^T x$, on the $d+1$-dimensional input feature space of x considering a *bias* term, where $w \in \mathbb{R}^{d+1}$. Subsequently, through a *sigmoid transformation*, Eq. 2.11, the output is converted into a number between 0 and 1, $s(w^T x)$. For each single data (feature) vector x_i, the output represents the probability of the given data to be classified as 1, Eq. 2.12. Based on a defined threshed, ϵ, usually 0.5, the predicted class label $\hat{y}_i$ is computed by a decision rule, Eq. 2.13.

$$s(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

$$p(y_i = 1|x_i) = s(w^T x_i) \tag{2.12}$$

$$\hat{y}_i = \begin{cases} 1 & s(w^T x_i) \geq \epsilon \\ 0 & otherwise \end{cases} \tag{2.13}$$

From Eq. 2.11, it is clear that the model relies on the mapping function $w^T x$. Thus, the objective of the LR is to find the optimal parameters w. As mentioned above, the cross-entropy loss can be one suitable loss function for optimizing the LR model parameters, reproduced in Eq. 2.14 by plugging Eq. 2.12 to represent the probabilities. It is noted that log function in Eq. 2.14 is the natural logarithm.

$$L(w) = \sum_{i=1}^{n} \left\{ y_i \log \left[\frac{1}{s(w^T x_i)} \right] + (1 - y_i) \log \left[\frac{1}{1 - s(w^T x_i)} \right] \right\} \tag{2.14}$$

Moreover, for each input x_i and ground truth y_i, Eq. 2.14 can be simplified to represent the loss for each single data point as follows:

$$L_i = \begin{cases} -\log(s(w^T x_i)) & y_i = 1 \\ -\log(1 - s(w^T x_i)) & y_i = 0 \end{cases} \tag{2.15}$$

From Eq. 2.15, if $s(w^T x_i) = 1$, there is zero loss penalizing the model implying that the model perfectly recognizes the class. However, it is a rare case. In practice, the probabilities obtained by the model are usually less than 1, which leads to produce non-zero values of the loss. However, this practical situation does not influence the optimization procedure because the trend is still the same, i.e., a higher probability always leads to a lower value of the loss.

One-hot vector encoding

Extending the LR from a binary classification to a multi-class setting, some modifications are needed. Firstly, *one-hot vector encoding* is used to represent the class label. Suppose the total number of classes is K, the i-th class is encoded as $[0, 0, \ldots, 1, \ldots, 0]^T$, which consists of 0's in all elements with the exception of a single 1 for the i-th entry. For example, if the total number of classes is 3, the label for the second class is encoded as $[0, 1, 0]^T$.

The weight representation of w is modified to a matrix form associated with every class, namely $W \in \mathbb{R}^{K\times(d+1)}$, where the k-th row of the W matrix is w_k^T, which is similar to that in binary case. Therefore, for $x_i \in \mathbb{R}^{d+1}$, the new mapping output is represented in Eq. 2.16, which is also known as the *logits*.

$$Wx_i = [w_1^T x_i, w_2^T x_i, \ldots, w_K^T x_i]^T = [z_1, z_2, \ldots, z_K]^T \tag{2.16}$$

Softmax function

The *softmax* function, Eq. 2.17, can be used to replace the sigmoid function. For x_i, the probability vector for all K classes is computed in Eq. 2.17. After the softmax operation, all entries are between 0 and 1 and sum up to 1, representing the probability distribution.

$$Softmax\left([z_1, z_2, \ldots, z_i, \ldots, z_K]\right) = \left[\frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}}, \frac{e^{z_2}}{\sum_{j=1}^{K} e^{z_j}}, \ldots, \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \ldots, \frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}}\right] \tag{2.17}$$

Based on one-hot encoding, both predictions and ground truth are encoded in the vector form, which actually act as the distributions from the perspective of statistics. Denote $\hat{P}$ as the target distribution from the ground truth y_i, and its label is class l among all the K classes. Thus, $P(\hat{P}_i = l) = y_i[l]$ represents the probability of the l-th class label ($y_i[l]$ is the l-th entry of the label vector y_i), which is 1 and the only non-zero entry in the $\hat{P}$, and the entire distribution is concentrated on the label. Similarly, denote Q as the estimated distribution from the above softmax function, the predicted probability of the l-th class is thus $P(Q_i = l) = e^{z_l}/\sum_{j=1}^{K} e^{z_j}$.

Kullback–Leibler (KL) divergence loss function

Unlike binary LR, the loss function in the multi-class case is evaluated by the difference between the target distribution $\hat{P}$ and the estimated distribution Q, which is computed by Kullback–Leibler (KL) divergence, $D_{KL}(\hat{P}||Q)$. Therefore, the optimal weight matrix W^* can be obtained by minimizing the KL divergence using all samples, as shown in Eq. 2.18.

$$
\begin{aligned}
W^* &= \underset{W}{\operatorname{argmin}}\, L(W) = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{n} D_{KL}(\hat{P}_i || Q_i) \\
&= \underset{W}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{K} P(\hat{P}_i = j) \log \left(\frac{P(\hat{P}_i = j)}{P(Q_i = j)} \right) \\
&= \underset{W}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{K} y_i[j] \log \left(\frac{y_i[j]}{softmax([w_1^T x_i, \ldots, w_K^T x_i])[j]} \right) \\
&= \underset{W}{\operatorname{argmin}} \left[- \sum_{i=1}^{n} \sum_{j=1}^{K} y_i[j] \log \left(\frac{e^{w_j^T x_i}}{\sum_{k=1}^{K} e^{w_k^T x_i}} \right) \right]
\end{aligned}
\tag{2.18}
$$

Note that in Eq. 2.18, if $\hat{P}_i = j$, $y_i[j] = 1$, and if $\hat{P}_i \neq j$, $y_i[j] = 0$, i.e., $y_i[j] = \delta_{\hat{P}_i,j}$ where $\delta_{\hat{P}_i,j}$ is the Kronecker delta. However, the computation of the cross-entropy loss and KL divergence loss in LR have no analytic closed-form solution. Thus, to minimize these loss functions, some algorithms, e.g., gradient descent, are used, which is introduced in a later section.

2.3 Support Vector Machine

The support vector machine (SVM) is one of the most popular ML algorithms, which is applicable to classification, regression, or other tasks. It was first developed by Vladimir Vapnik and co-workers at AT&T Bell Laboratories in 1995 [21], and has received considerable attention in both industry applications and research endeavors.

SVM falls into the scope of supervised learning. Through constructing a *hyper-plane* or a set of hyper-planes in a high dimensional space, the data can become separable by these planes based on their class labels. The hyper-plane is also known as *decision boundary*, H, and the data points closest to H are denoted as the *support vector* (SV) *points* and the minimum distance from the decision boundary H to any of the SV points are known as the *margin*, M. Herein, H is denoted as $H = \{w^T x - b = 0\}$, where the vector w is perpendicular to the hyper-plane and b is the offset of the hyper-plane from the origin along the normal vector w.

Similar to the LR discussion, starting from the binary classification, the setup of the problem is as follows: given a training set $D = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and y_i is labeled by either -1 or $+1$ (two classes), the SVM is to find a $d-1$-dimensional hyper-plane (decision boundary) H to separate the two classes. In general, a good separation is achieved by a large margin, which indicates a lower generalization error of the classifier.

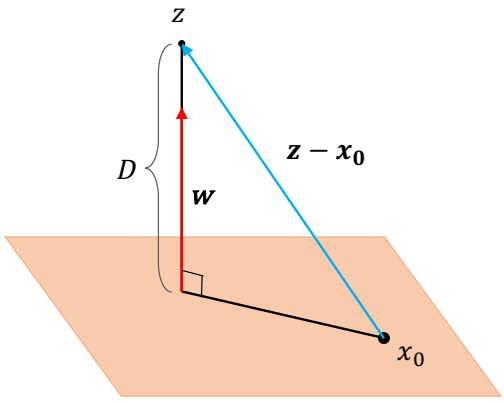


Figure 2.4: Illustration of the distance D by projecting $(z - x_0)$ on w

Hard-margin SVM

In the hard margin SVM, the classifier is trained to find a hyper-plane H that maximizes the margin M and subjected to several constraints: (1) all points classified as $+1$ are on the positive side of H and the distance to H is larger than M; (2) all points classified as -1 are on the negative side of H and the distance to H is also larger than M; and (3) the value of the margin M is non-negative.

From Fig. 2.4, taking one arbitrary point z, and another arbitrary point x_0 on the hyper-plane H, i.e., $w^T x_0 = b$, the distance D from z to H is the length of the projection of the vector $z - x_0$ to the vector w (perpendicular to H) obtained by the dot product $w^T(z - x_0)$, which is normalized by the length of the vector w, i.e., its Euclidean ($\mathcal{L}_2$) norm $\|w\|_2$, Eq. 2.19. Thus, the distance from any points x_i to H in the training set is $(w^T x_i - b)/\|w\|_2$.

$$D = \frac{w^T(z - x_0)}{\|w\|_2} = \frac{w^T z - w^T x_0}{\|w\|_2} = \frac{w^T z - b}{\|w\|_2} \tag{2.19}$$

With the representation of D and considering the above-mentioned three constraints, the optimization problem can be formulated in Eq. 2.20.

$$\begin{aligned} \max_{w,b,M} \quad & M \\ \text{s.t.} \quad & y_i \frac{w^T x_i - b}{\|w\|_2} \geq M \qquad \forall i \\ & M \geq 0 \end{aligned} \tag{2.20}$$

To eliminate the variable M and for further simplification, new variables $w' = \frac{w}{\|w\|_2 M}$ and $b' = \frac{b}{\|w\|_2 M}$ are introduced, and now Eq. 2.20 is equivalent to:

$$\begin{aligned} \max_{w',b'} \quad & \frac{1}{\|w'\|_2} \\ \text{s.t.} \quad & y_i(w'^T x_i - b') \geq 1 \qquad \forall i \end{aligned} \tag{2.21}$$

In the domain of ML and optimization, minimizing loss is more preferable than maximizing gain. To make the objective function more precise and its solution more numerically stable and robust in the computations, Eq. 2.21 can be rewritten in a minimization format along with renaming the variables, squaring the objective function, and scaling with a constant multiplier $\frac{1}{2}$, as follows,

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_i(w^T x_i - b) \geq 1 \qquad \forall i \end{aligned} \tag{2.22}$$

Finally, the objective of hard-margin SVM is to solve the optimization problem in Eq. 2.22. It is noted that the norm in Eq. 2.22 can be taken other than the Euclidean one. Therefore, the superscript of the used norm is dropped in Eq. 2.22 and also in the sequel for generality.

Soft-margin SVM

In real applications, the available data is usually not linearly separable. Accordingly, the hard-margin SVM may have no solution. Moreover, hard-margin SVM is sensitive to the outliers, which makes the classifier less generalized. To address these two issues, soft-margin SVM is introduced. Through modifying the constraints of the hard-margin SVM using *slack variable* ξ (denote ξ_i as one realization of ξ for the i-th data point, which is a scalar and assigned a specific optimal value of the slack variable, as derived in the next subsection), some data points can now violate the constraints, by actually constructing a less-strict version of the hard-margin SVM. Therefore, the new objective is established in Eq. 2.23 as follows,

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(w^T x_i - b) \geq 1 - \xi_i \qquad \forall i \\ & \xi_i \geq 0 \qquad \forall i \end{aligned} \tag{2.23}$$

where C is a hyper-parameter, which needs to be pre-determined by the user. It is usually firstly determined by prior knowledge and then tuned by cross-validation.

Dual SVM & kernel trick

Instead of directly solving Eq. 2.23, it is more desirable to reformat it and solve its dual optimization problem. This is because (1) the dual problem transfers the inequality constraints into equality constraints; (2) it is easier to introduce the so-called "kernel trick" to solve

nonlinearly separable data; and (3) the dual problem changes the complexity of the problem, where the primal problem is related to the dimension of the feature vector w, i.e., d, but the dual problem is related to number of the data, i.e., n.

Dual SVM

First, the primal and dual variables are defined for solving the SVM problem. The primal variables are w, b, and ξ for the constraints in Eq. 2.23. On the other hand, the introduced dual variables are α and β, whose realizations for the i-th data point are α_i and β_i, respectively. Therefore, the *Lagrangian* for the soft-margin SVM is as follows,

$$\begin{aligned} L(w, b, \xi, \alpha, \beta) &= \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i\left((1-\xi_i) - y_i(w^T x_i - b)\right) + \sum_{i=1}^{n}\beta_i(-\xi_i) \\ &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i y_i(w^T x_i - b) + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}(C - \alpha_i - \beta_i)\xi_i \end{aligned} \tag{2.24}$$

Thus, the dual problem is stated as follows,

$$\max_{\alpha,\beta\geq 0} g(\alpha, \beta) = \min_{w,b,\xi} \frac{1}{2}||w||^2 - \sum_{i=1}^{n}\alpha_i y_i(w^T x_i - b) + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}(C - \alpha_i - \beta_i)\xi_i \tag{2.25}$$

The optimal dual variables can be obtained by using *Karush–Kuhn–Tucker* (KKT) *conditions*. For the stationarity conditions of KKT, the optimal primal values should satisfy the following:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial \xi_i} = 0 \tag{2.26}$$

According to the three conditions in Eq. 2.26, the optimal values are related as follows: $w^* = \sum_{i=1}^{n}\alpha_i^* y_i x_i$, $\sum_{i=1}^{n}\alpha_i^* y_i = 0$, and $C - \alpha_i^* - \beta_i^* = 0$, which means that $0 \leq \alpha_i^* \leq C$.

Replacing the variables with optimal ones in Eq. 2.24, a more compact form of the objective function is obtained as follows,

$$L(w, b, \xi, \alpha^*, \beta^*) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n}\alpha_i^* y_i w^T x_i + \sum_{i=1}^{n}\alpha_i^* \tag{2.27}$$

Moreover,

$$\begin{aligned} g(\alpha^*, \beta^*) &= \min_{w,b,\xi} L(w, b, \xi, \alpha^*, \beta^*) \\ &= L(w^*, b^*, \xi^*, \alpha^*, \beta^*) \\ &= \frac{1}{2} \left|\left| \sum_{i=1}^{n} \alpha_i^* y_i x_i \right|\right|^2 - \sum_{i=1}^{n} \left(\alpha_i^* y_i x_i^T \left(\sum_{j=1}^{n} \alpha_j^* y_j x_j \right) \right) + \sum_{i=1}^{n} \alpha_i^* \\ &= \sum_{i=1}^{n} \alpha_i^* - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i^* y_i x_i^T x_j y_j \alpha_j^* \end{aligned} \tag{2.28}$$

Finally, the dual optimization problem can be defined as follows,

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i x_i^T x_j y_j \alpha_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \quad i = 1, \ldots, n \end{aligned} \tag{2.29}$$

Once the optimal solution of $\alpha^* = (\alpha_1^*, \alpha_2^*, \ldots, \alpha_i^*, \ldots, \alpha_n^*)^T$ is obtained, plug it back into the above three conditions leading to the optimal values to determine the optimal primal variables w^* (Eq. 2.30), b^* (Eq. 2.31), and the specific choices of ξ_i (Eq. 2.32).

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i \tag{2.30}$$

$$b^* = y_j - w^{*T} x_j = y_j - \sum_{i=1}^{n} \alpha_i^* y_i x_i^T x_j \tag{2.31}$$

$$\xi_i^* = \begin{cases} 1 - y_i(w^{*T} x_i - b^*) & \alpha^* = C \\ 0 & otherwise \end{cases} \tag{2.32}$$

where b^* is obtained using one single data point (x_j, y_j) with α_j^* that satisfies the condition of $0 \leq \alpha_j^* \leq C$. In addition, b^* can be taken as the mean of all points whose $\alpha_j^{*\prime}s$ satisfy the condition.

With the known values of w^* and b^*, the hyper-plane is also known from the following:

$$w^* x + b^* = 0 \tag{2.33}$$

Therefore, the prediction $\hat{y}$ for any input data x is as follows,

$$\hat{y} = sgn(w^* x + b^*) \tag{2.34}$$

where $sgn(.)$ function is defined as follows,

$$sgn(z) = \begin{cases} +1 & z > 0 \\ 0 & z = 0 \\ -1 & z < 0 \end{cases} \tag{2.35}$$

Kernel trick

The above formation of SVM is based on the assumption that the data is linearly separable; however, in reality this may be difficult to satisfy. Moreover, the vector/matrix computation in the above formulation is not efficient. Therefore, the so-called "kernel trick" using kernel function is introduced and found to be effective to address these issues.

Using a linear SVM classifier in the raw data (feature) space is usually very limited. However, if some mapping (transformation) is performed on the raw data, SVM may still work effectively in the augmented feature space. Suppose an arbitrary mapping function, $\phi(.)$ can be found such that it can elevate the raw data (feature) space χ to a higher dimensional augmented feature space $\mathbb{H}$. Therefore, the transformed data $\phi(x)$ in the augmented space may become linearly separable. However, it is not easy to find such mapping function.

In the "kernel trick", a kernel function $k(x, z)$ is defined between two inputs, x and z. It contains the inner product of the implicit mapping $\varphi(.)$ between these two inputs. Assume there exists a mapping from χ to $\mathbb{H}$, the "kernel trick" leads to an approximation of the inner product of the nonlinear mapping $\phi(.)$ as follows: $k(x, z) = \varphi(x) \cdot \varphi(z) \approx \phi(x) \cdot \phi(z)$. With the defined kernel function, the user can conduct computations directly instead of explicitly applying the mapping function $\phi(.)$. In addition, using the "kernel trick" reduces the computational complexity where using a kernelized method is proven to be more efficient than using a non-kernelized method for some cases.

Now, we go back to solving the SVM optimization problem. First, changing Eq. 2.29 to a vector form, the objective function can be rewritten as shown in Eq. 2.36.

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i x_i^T x_j y_j \alpha_j = \alpha^{*T} \mathbf{1} - \frac{1}{2} \alpha^{*T} Q \alpha^* \tag{2.36}$$

where $\mathbf{1}$ is the unit vector whose all entries are 1 with the same dimension of α^*, and $Q_{ij} = y_i x_i^T x_j y_j$.

From Eqs 2.29 & 2.36, the only operation related to the data x is the inner product, which is convenient to directly apply the "kernel trick". Through constructing the kernel function $k(.,.)$, $k(x_i, x_j)$ is computed instead of the the original inner product. The full mapping matrix from the raw input data $X = [x_1, x_2, \ldots, x_n]^T$ to the augmented feature space $\mathbb{H}$ is called the *Gram matrix* and is expressed as follows,

$$K(X,X) = \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{pmatrix} \tag{2.37}$$

A valid kernel function should be positive semi-definite (PSD), which is difficult to validate using an arbitrary limited training dataset. Thus, in practice, several empirical kernel functions are usually used, e.g., linear kernel (Eq. 2.38), polynomial kernel (Eq. 2.39) and Gaussian kernel (Eq. 2.40), also known as the radial basis function (RBF) kernel.

$$k(x,z) = x \cdot z + c \tag{2.38}$$

where c is an optional constant.

$$k(x,z) = (\alpha(x \cdot z) + c)^p \tag{2.39}$$

where the adjustable parameters are slope α, constant c, and polynomial degree p, respectively.

$$k(x,z) = \exp(-\frac{\|x-z\|^2}{2\sigma^2}) \tag{2.40}$$

where the adjustable parameter σ plays a major role in the performance of the kernel, and should be carefully tuned to the problem at hand.

2.4 Decision Tree & Random Forest

Decision tree (DT) and random forest (RF) are two of the most popular ML algorithms and have been widely applied in both classification and regression problems. Similarly, DT and briefly RF are introduced in this dissertation due to the increased demand in SHM to use ML for solving several classification problems.

DT model

The classification DT forms a tree-like classifier (model), which consists of nodes and directed edges. The type of the node is either an *internal node* or a *leaf node*, where the former represents the feature and the latter represents the class label. Starting from the root node of the tree, the classifier recursively keeps partitioning the input space by nodes based on certain criteria until arriving at the leaf node. Finally, the input data space is separated into several regions.

Suppose the input data x has 3 features, namely x^1, x^2, and x^3, and the target output describes two states of a building structure, namely *damaged state* and *undamaged state*. Fig. 2.5 illustrates the procedure of how the DT works. Starting from the root node, if the

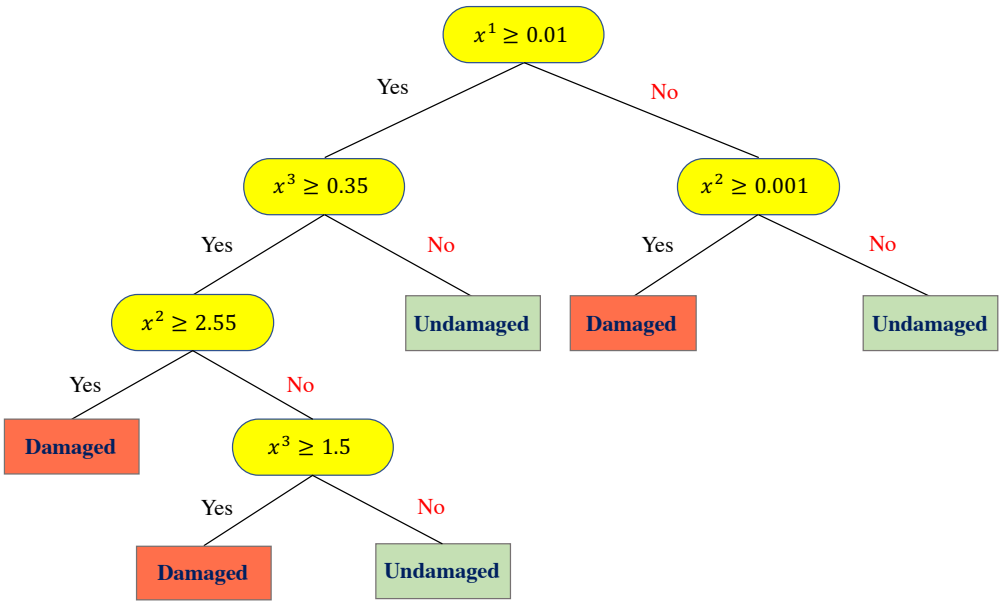


Figure 2.5: Illustration of a hypothetical DT model for three-dimensional data

first feature of the data satisfies the condition of $x^1 \geq 0.01$, the data is further examined in the next node by the third feature x^3 on the left branch; otherwise, it is examined by the condition of second feature x^2 on the right branch. The rectangle boxes represent the leaf nodes, and the intersection of all conditions from the root node to one leaf node forms one region for a particular class label. For example, the edges in the left-most branch form one decision region for the data labeled as damaged state, where features are satisfying the conditions of $\{x^1 \geq 0.01\} \cap \{x^3 \geq 0.35\} \cap \{x^2 \geq 2.55\}$. Exploring the tree branches by all the data, the full DT model is established and the regions for both class labels are also determined. Once the new data comes into the model, through matching the features with the conditions and the determined regions, the predicted class label is obtained.

Feature selection

As mentioned above, in each node, DT actually works with a feature-value pair, (x^i, v), and data is separated into two classes through examining whether the feature x^i exceeds the threshold value v or not. According to the example shown in Fig. 2.5, the split results may be influenced by how the features x^i are selected and split. Therefore, multiple tree models can lead to similar separation results where a good feature selection criterion is required. The *information gain* is one of the most commonly used criteria in DT.

Information gain

For convenience and readability, some basics related to *entropy* are introduced herein. In the information theory and statistics, entropy is a measure that represents the uncertainty of the

random variable. Suppose a random variable X is discrete, which has N possible outcomes, i.e., x_1, x_2, ..., x_N (not to be confused with the data points expressed in Eq. 2.2, and its distribution is as follows,

$$P(X = x_i) = p_i \quad i = 1, 2, \ldots, N \tag{2.41}$$

where p_i equals to the ratio between the number of data belong to outcome x_i and number of total data.

The entropy of X, denoted as $H(X)$, is expressed by Eq. 2.42 where here the base of the log function is 2, which is the convention in the *information theory*.

$$H(X) = -\sum_{i=1}^{N} p_i \log(p_i) \tag{2.42}$$

Suppose there exist two random variables X and Y, and their joint probability distribution is known as follows,

$$P(X = x_i, Y = y_j) = p_{ij} \quad i = 1, 2, \ldots, N;\ j = 1, 2, \ldots, M \tag{2.43}$$

The *conditional entropy* $H(X|Y)$, which represents the uncertainty of the variable X under the condition of knowing the variable Y, is expressed as follows,

$$H(X|Y) = -\sum_{j=1}^{M} P(Y = y_j) H(X|Y = y_j) \tag{2.44}$$

If the probabilities used in the entropy and the conditional entropy are obtained by statistical estimation based on data, they can be called the *empirical entropy* and the *empirical condition entropy*, respectively.

The above-mentioned information gain, $IG(X; Y)$, which is also known as the *mutual information*, is defined as follows,

$$IG(X; Y) = H(X) - H(X|Y) \tag{2.45}$$

Under the working principle of the DT, the objective of each node can be thought of as to maximize the information gain after the feature-splitting operation, where Y in Eq. 2.45 can represent such split event of operation. Take as an example of feature split of the root node shown in the Fig. 2.5, where Y has two possible outcomes, i.e., $M = 2$ in Eqs 2.43 & 2.44, namely $\{x^1 \geq 0.01\}$ and $\{x^1 < 0.01\}$. These outcomes can further be represented by $Y = 1$ and $Y = 0$, respectively. Thus, the information gain after the split in the root node is presented as follows,

$$\begin{aligned} IG(X; Y) &= H(X) - \big(P(x^1 \geq 0.01) H(X|x^1 \geq 0.01) + P(x^1 < 0.01) H(X|x^1 < 0.01)\big) \\ &= H(X) - (P(Y = 1) H(X|Y = 1) + P(Y = 0) H(X|Y = 0)) \end{aligned} \tag{2.46}$$

Similarly, based on Eq. 2.46, the calculation of the information gain continues in a *greedy* way (i.e., making locally optimal choice at each stage with the intent of finding a global optimum), where each step results in the maximum information gain, until the computation reaches the leaf nodes or designated depth in the DT defined by the user. In addition, the DT using the above approach for information gain computation is called ID3 (Iterative Dichotomize 3) [96], but it only works for the dataset with discrete features.

Gini index

Based on different ways of computing the information gain and conducting feature selection, there are different types of DT algorithms other than ID3, e.g., C4.5 [95], and classification and regression tree (CART) [11], where C4.5 overcomes the shortcomings of ID3, and CART is a more general algorithm. It is worth noting that the information gain in CART is alternatively computed by the *Gini index*, which is also a measurement of the information uncertainty. Gini index is also known as *Gini Impurity*.

Suppose we have a K-classes classification problem, and the probability of the data which belongs to the k-th class is denoted as $p_k = |l_k|/|CT|$, computed by the ratio between the number of data labeled for the k-th class, $|l_k|$, and the number of data in this collection, $|CT|$. Thus, the Gini index (developed by Corrado Gini in 1912 [35]) is calculated as follows,

$$Gini(CT) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2 \tag{2.47}$$

Let the selected feature to be A and the collection CT can be separated into I parts by a certain threshold of the feature, namely CT_1,..., CT_I. Thus, the conditional Gini index is computed as follows,

$$Gini(CT|A) = \sum_{i=1}^{I} p_i Gini(CT_i) = \sum_{i=1}^{I} \frac{|l_i|}{|CT_i|} Gini(CT_i) \tag{2.48}$$

The information gain using the Gini index under the condition of the feature A is presented in Eq. 2.49, which is also known as the *Gini information gain*.

$$Gini\ IG = Gini(CT) - Gini(CT|A) \tag{2.49}$$

For each node, the Gini information gain is computed for all features and the one that has the largest gain is selected as the split feature in the current node. Compared with the above-mentioned entropy-based information gain, using Gini index may achieve better computational efficiency than using entropy due to waiving the logarithm operations.

RF Model

The above-mentioned DT only represents one single tree, which may be limited or suffer from over-fitting issues. Intuitively, if multiple trees can be utilized at the same time, they may

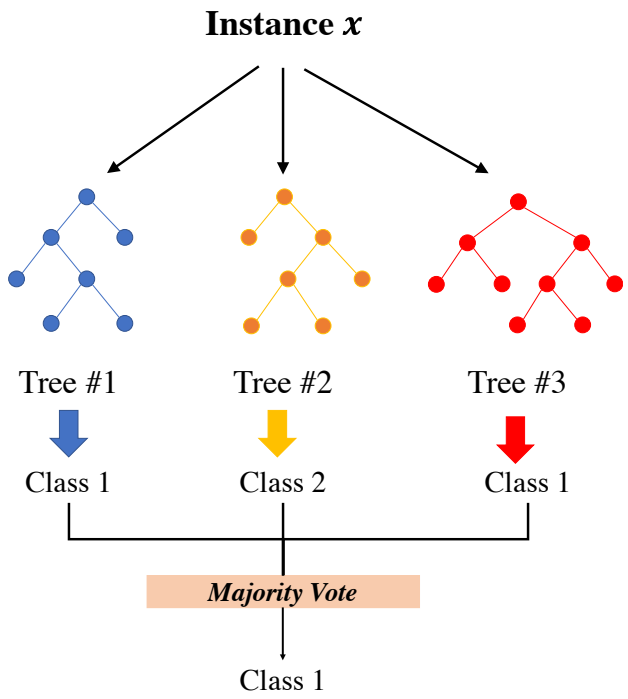


Figure 2.6: Prediction procedure of the RF

yield more robust results. As a result, random forest (RF) is proposed by Leo Breiman [10], which is an *ensemble learning* algorithm that performs predictions by using multiple DT's, and the output can be the mode of the classes in a classification problem or the mean of the prediction in a regression problem from individual trees.

There are many ways to conduct RF computations according to many past studies. This brief subsection only introduces one conventional method, namely *bagging*, which also stands for *bootstrap aggregating*. While conducting bagging, the major steps are as follows: (1) construct subsets with size m from raw dataset whose size is n, (2) train T weak models (classifier or regressor) on the subsets, and (3) assemble single models for prediction, Fig. 2.6. For example, suppose $T = 50$ DT's are used in the RF, and the number of the total data in the dataset is $n = 500$. Firstly, 50 subsets are constructed through randomly sampling with replacement from the raw dataset, and the size of the subsets is, e.g., $m = 400$. Then, these 50 DT's are trained on the different subsets of size 400. While in prediction, the test data is passed to all DT's, and there will be 50 outputs from the 50 trees. For a classification problem, the final decision on the class label is the majority vote among all outputs, e.g., if for the $T = 50$, 47 trees predict the building is undamaged, but the remaining 3 trees predict it is damaged, thus the final prediction is undamaged.

The bagging method treats each tree equally, and there are many other ways to improve it for better performance. These other ways yield different RF methods, e.g., Boosting, AdaBoost, etc. More details can be found in references [9, 66].

Chapter 3

Basics of Deep Learning

As a subset of machine learning (ML), deep learning (DL) is attracting more interest nowadays. However, the study of DL dates back to 1940s, and it has undergone multiple periods of research and development [38]. The concept of DL originated from the study in neuroscience, and its objective is to simulate the mechanisms of human brain to understand and interpret the data, e.g., images, texts, and sounds.

A DL model is thought of as a composition of multiple levels of functions, and is usually represented by stacking multiple layers which consist of special units, i.e., neurons. Currently, *neural network* (NN), *convolutional neural network* (CNN), *recurrent neural network* (RNN), and *long short-term memory* (LSTM) *network* are several common realization methods of DL. Through stacking layers to a certain depth, the number of model parameters increases. As a result, the model gains a significant representation learning ability to learn complex patterns of the data. The word "deep" in the name of DL partially refers to this characteristic.

In this chapter, some fundamental knowledge of the DL including NN and CNN are introduced in the subsequent sections. For more details, the reader is referred to [38].

3.1 Neural Network

From the long history of development, there are many ways to name the NN, e.g., artificial NN (ANN), feed-forward NN, fully-connected (fc-) NN, and multi-layer perceptron (MLP). Actually, by detailed definition, the above-mentioned names may have slight differences in modeling, but nowadays researchers are usually unifying them into NN with the following characteristics: (1) have at least one layer, (2) all neurons between layers are linked together, but are unconnected within the same layer, and (3) all connections are feed-forward and have no recurrence. Through a large number of connected neurons activated by certain nonlinear functions, NN gains a powerful representation ability, which can model the working mechanism of a biological brain to process the input information and to realize *artificial intelligence* (AI). To provide theoretical background for the approximation ability of the NN, *universal approximation theorem* was proposed by Kurt Hornik [45], which states that:

"A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions". It also states that: "Simple neural networks can represent a wide variety of interesting functions when given appropriate parameters." Thus, instead of using one layer, NN is extended from the theory to the current practice and research where multi-layer NN is adopted, i.e., deep NN, which is thought to have capability to well-approximate more complex functions or patterns.

To obtain the best approximation, the large number of parameters in the neurons are trained with the input data through several iterations with the objective of minimizing the defined *loss function*. Usually, the NN is trained iteratively, and the workflow in NN's consists of two passes: (1) forward pass, and (2) backward pass. A standard architecture of the NN usually consists of three parts: (1) an input layer, (2) multiple hidden layers, and (3) an output layer. An example of a three-layer NN (or two hidden layer NN) is illustrated in Fig. 3.1. Note that the input layer is typically not counted in the designation of the NN.

Forward propagation

For supervised learning problems, the forward pass between layers can be summarized as:

1. Flatten the input (features) to a row vector as the input for the input layer.
2. Multiply entries of the vector by weights on the edges between nodes in two different layers.
3. Sum over weighted entries pointing to the node and also add a bias term for each node in the next layer.
4. Activate the results obtained in (3) via the activation functions to produce outputs, which are treated as the inputs for the next layer.
5. Repeat steps (1) to (4) until the forward pass arrives to the last output layer, where the outputs usually represent probability values.
6. Compute the loss by the outputs in (5) and the ground truth (labels).

For a multi-class classification problem, suppose the input data/label tensor pair is $(\mathbf{x}, \mathbf{y})$, and assume the NN has a total of M layers. For example, the data tensor $\mathbf{x} \in \mathbb{R}^{N\times W\times H\times C}$ can be from stacking N images, $\{x_1, x_2, ..., x_N\}$, where each image has width W pixels, height H pixels and color channels C, e.g., $C = 1$ for the gray scale in black and white images, $C = 3$ for RGB (Red, Green and Blue) channels in colored images, or $C = 4$ for RGB channels + a depth channel for colored images taken with a depth camera. The label tensor $\mathbf{y} \in \mathbb{R}^{N\times K}$ is stacked by $\{y_1, y_2, ..., y_N\}$ after one-hot encoding according to a total of K classes. Denote the input of the l-th layer as $\mathbf{I}^{[l]}$, the multiplication operation and subsequent activation can be represented as functions $f^{[l]}(\mathbf{I}^{[l]}) = (\mathbf{W}^{[l]})^T\mathbf{I}^{[l]}$ and $\sigma^{[l]}(.)$, respectively. The output of $f^{[l]}(\cdot)$ is denoted as $\mathbf{z}^{[l]}$. In its basic form, the multiplication operation, according to Fig. 3.1,

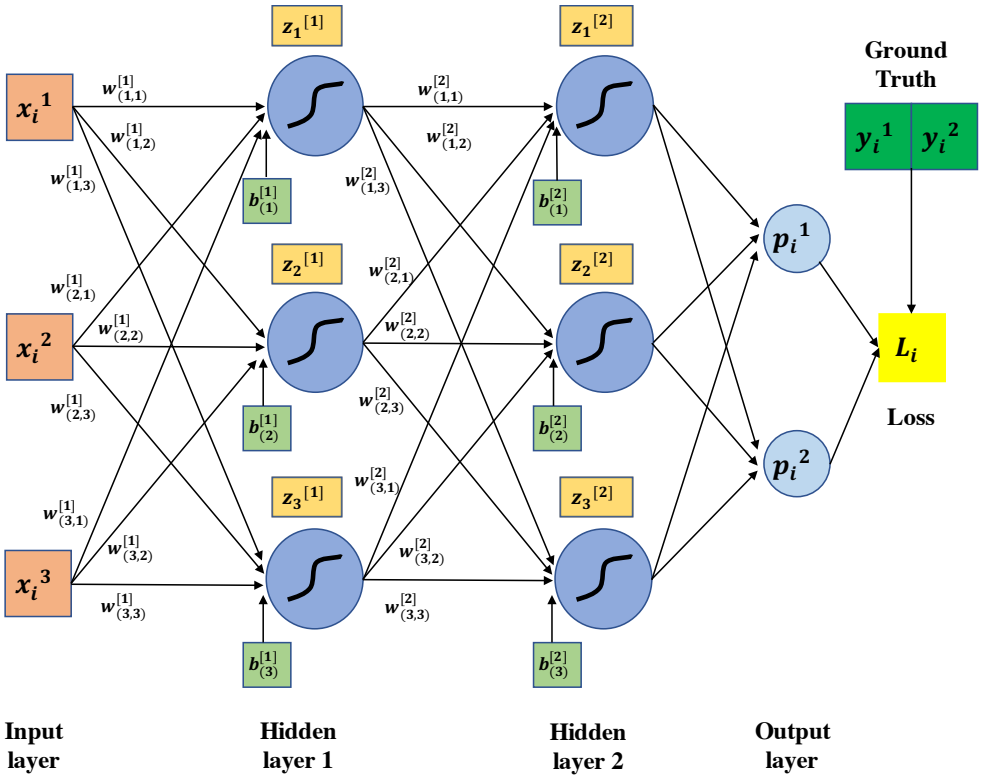


Figure 3.1: Example of a three-layer NN

performs the computation for single neuron unit: $z = \mathbf{w}^T\mathbf{x} + b$ (e.g., $z_1^{[1]} = \mathbf{w}^T\mathbf{x} + b_{(1)}^{[1]}$ for the first neuron in the hidden layer 1.), where $\mathbf{w}$ & $\mathbf{x}$ can be augmented to include b (the bias term) and the constant 1 in an additional feature channel (dimension), respectively, to give $\mathbf{W}$ & $\mathbf{I}$. Here, z is the input to the activation function to produce the output of a particular hidden layer, which is the input to the subsequent one. Some common activation functions $\sigma^{[l]}(.)$ are *sigmoid* (Eq. 3.1), *rectified linear unit* (ReLU) [89] (Eq. 3.2), *leaky ReLU* [74] (Eq. 3.3), softmax (Eq. 2.17), and *hyperbolic tangent* (*tanh*) (Eq. 3.4), where ReLU is commonly adopted in most layers and sigmoid and softmax are usually applied in the last layer to transfer the output into probability values.

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1} \tag{3.1}$$

$$\sigma(z) = max(0, z) \tag{3.2}$$

$$\sigma(z) = \begin{cases} z & z \geq 0 \\ \alpha z & z < 0 \end{cases} \tag{3.3}$$

where $\alpha > 0$ is the absolute value of the negative slope parameter and usually $\alpha = 0.01$.

$$\sigma(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.4}$$

The output after activation is the so-called feature, $\mathbf{A}^{[l]}$, where $\mathbf{A}^{[l]} = \sigma^{[l]}(f^{[l]}(\mathbf{I}^{[l]}))$ and is used as the input for the $(l+1)$-th layer, i.e., $\mathbf{I}^{[l+1]} = \mathbf{A}^{[l]}$. The forward activation pass propagates through the entire network until the signals reach the last layer. The whole procedure can be heavily parametrized by a combination of function composition, Eq. 3.5, denoted as $s(.)$, where $s(\mathbf{x}) \in \mathbb{R}^{N \times K}$ if feeding data $\mathbf{x}$ corresponds to the K classes defined in $\mathbf{y}$. Then, the total loss of the NN can be computed based on the defined loss function $L(.)$, refer to Section 2.1, with $s(\mathbf{x})$ and label $\mathbf{y}$, Eq. 3.6.

$$s(\mathbf{x}) = \sigma^{[M]}\left(f^{[M]}\left(\sigma^{[M-1]}\left(f^{[M-1]}\left(\dots f^{[2]}\left(\sigma^{[1]}\left(f^{[1]}\left(\mathbf{x}\right)\right)\right)\dots\right)\right)\right)\right) \tag{3.5}$$

$$L = L\left(s(\mathbf{x}), \mathbf{y}\right) \tag{3.6}$$

Backward propagation

The training objective of the NN is to minimize the loss and to find the optimal parameters in the layers, which are denoted as $\mathbf{W}$. However, the optimization in the NN is usually *non-convex* and the optimization problem is difficult to be solved through conventional algorithms. As a result, a gradient-based updating algorithm, namely *gradient descent* [38], was developed by updating the model parameters with the gradient of the loss with respect to these parameters in a *greedy* way, and finally arrive at relative "good" local optima. It is worth noting that it is impractical to obtain the global optima in most cases of the NN's. However, usually some local optima are equivalently as good as the global optima. Thus, it is reasonable and acceptable to use these "good" local optima as the way to practically terminate the NN training cycles.

The gradient descent algorithm is usually summarized in the following steps:

1. Initialize parameters randomly or from certain distributions, e.g., Gaussian distribution.
2. Update the parameters iteratively by subtracting the incremental change of the parameters in terms of the product of the gradient of loss with respect to the parameters and a user-defined scalar step η.
3. Repeat step(2) until the loss converges to a minimum or the number of iterations reaches a maximum designated value.

Take an example of an updating procedure of the parameters in the l-th layer, denoted as $\mathbf{W}^{[l]}$. The gradient of loss with respect to $\mathbf{W}^{[l]}$, $\nabla_{\mathbf{W}^{[l]}} L$ or $(\frac{\partial L}{\partial \mathbf{W}^{[l]}})^T$, is computed using the *chain rule* as follows:

$$
\begin{aligned}
\nabla_{\mathbf{W}^{[l]}} L &= (\frac{\partial L}{\partial \mathbf{W}^{[l]}})^T \\
&= (\frac{\partial L}{\partial \mathbf{A}^{[M]}} \cdot \frac{\partial \mathbf{A}^{[M]}}{\partial \mathbf{z}^{[M]}} \cdot \frac{\partial \mathbf{z}^{[M]}}{\partial \mathbf{A}^{[M-1]}} \cdots \frac{\partial \mathbf{z}^{[l+1]}}{\partial \mathbf{A}^{[l]}} \cdot \frac{\partial \mathbf{A}^{[l]}}{\partial \mathbf{z}^{[l]}} \cdot \frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{W}^{[l]}})^T \\
&= (\frac{\partial L}{\partial \mathbf{A}^{[M]}} \cdot (\sigma^{[M]})' \cdot (f^{[M]})' \cdot (\sigma^{[M-1]})' \ldots (\sigma^{[l]})' \cdot (f^{[l]})' \cdot \mathbf{I}^{[l]})^T \\
&= (\mathbf{I}^{[l]})^T \cdot (f^{[l]})' \cdot (\sigma^{[l]})' \ldots (\sigma^{[M-1]})' \cdot (f^{[M]})' \cdot (\sigma^{[M]})' \cdot \nabla_{\mathbf{A}^{[M]}} L
\end{aligned}
\tag{3.7}
$$

In each iteration step, the parameters $\mathbf{W}^{[l]}$ are updated by the gradients with the scale learning rate η, Eq. 3.8.

$$
\mathbf{W}^{[l]} \leftarrow \mathbf{W}^{[l]} - \eta \cdot \nabla_{\mathbf{W}^{[l]}} L \tag{3.8}
$$

It is noted that the presented approach in Eq. 3.7 to calculate the gradients with respect to the parameters is known as the *backpropagation* (BP) algorithm [38].

Stochastic gradient descent

Due to possible inefficiencies and limitations of the available computational resources, instead of obtaining the *full gradient* based on Eq. 3.7 (also known as the *batch gradient descent*, which is usually effective for *convex* problems), the *stochastic gradient descent* (SGD) method was proposed and now widely adopted [38]. In SGD, instead of using all the data at one time, in each iteration, the NN is only updated according to one small batch containing m ($m < N$) training points sampled from the full dataset of size N. More specifically, if $m = 1$, it is called SGD; otherwise it is called *mini-batch gradient descent* (MBGD). Using one sample per iteration in the SGD often leads to a noisy gradient. On the contrary, the MBGD is more stable. Moreover, the MBGD takes advantages of parallel computing on the vector/matrix form of the batch of m training points, which is more efficient than the similar computation in the SGD but needs m iterations. For brevity, in some conventional literature, both SGD and MBGD are simply unified as SGD with a specified batch size m.

In the SGD, the gradient in the k-th iteration is denoted as $\mathbf{G}_k$, which shares the same dimension as the parameters, $(\mathbf{W}^{[l]})^{(k)}$, and computed as follows,

$$
\mathbf{G}_k = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\mathbf{W}^{[l]}} L(s(x_i), y_i) \tag{3.9}
$$

Accordingly, the new updating rule in the k-th iteration is shown in Eq. 3.10. In the SGD, the learning rate $\eta^{(k)}$ is usually changeable during the iterations based on the learning history, and the configuration about its adjustment is defined as the *learning schedule*.

$$
(\mathbf{W}^{[l]})^{(k+1)} \leftarrow (\mathbf{W}^{[l]})^{(k)} - \eta^{(k)} \mathbf{G}_k \tag{3.10}
$$

It is noted that $\mathbf{G}_k$ is a random variable and its expectation satisfies $E[\mathbf{G}_k] = (\nabla_{\mathbf{W}^{[l]}} L)^{(k)}$. Thus, the gradient obtained in each iteration is an unbiased estimate of the true gradient, and the final result is thought to be close to the one using true gradient when making more and sufficient number of iterations. Moreover, in the SGD, instead of randomly sampling m training points, it is suggested to use the term *epoch* to conduct training in a more well-organized order, where one training epoch means to go over all the training data once [38]. At the beginning of each epoch, the sequence of the data fed into the NN is shuffled randomly, and then conducting the SGD batch by batch followed by this sequence until looping over all data points, where the batch size maintains the m value except for the last batch size, m_{last} with $m_{last} \leq m$. Shuffling and repeating continue until the NN loss converges or the designated number of training epochs is reached. In addition, the number of total iterations, N_{iter}, in this case for all epochs can be easily computed based on the number of all the data points N, the number of the training epochs N_{epoch}, and the batch size m as follows,

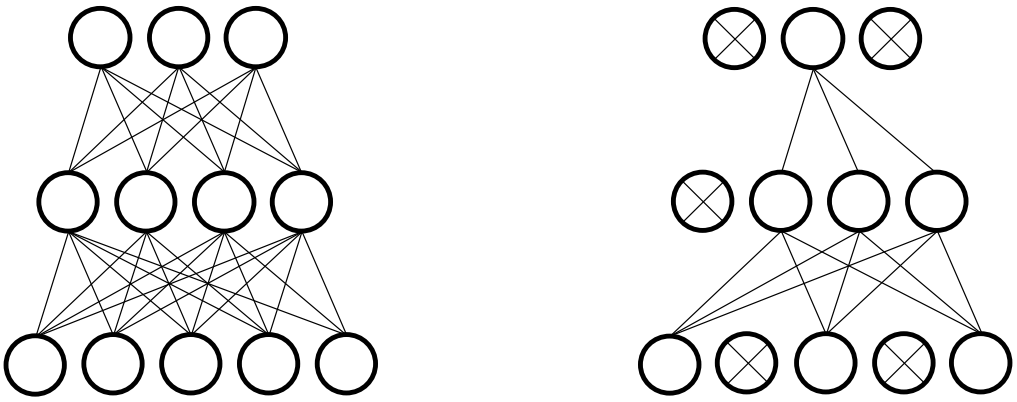
$$N_{iter} = N_{epoch} \cdot \left\lceil \frac{N}{m} \right\rceil \tag{3.11}$$

where $\lceil . \rceil$ is the ceiling operation such that it maps $\lceil x \rceil$ with $x \in \mathbb{R}$ to the least integer $\geq x$.

Dropout

When using more neurons and stacking more layers, the NN becomes a deep structure, namely deep NN, and it obtains a more powerful representation ability through a significantly increased number of parameters in the NN. However, when dealing with small datasets, using a deep NN is likely to trigger over-fitting issues. To address this, *dropout* is one of the possible choices, which was shown to be very efficient for relieving over-fitting in deep NN's [120]. The key idea of dropout is to randomly drop neurons along with connections between layers during training, and the dropped ones are not activated in both forward and backward propagation. However, during the test process, the original complete network is used, refer to Fig. 3.2. It is noted that the dropout is only performed in every training epoch, and the selection for the dropped connection is random and independent of the previous epoch.

According to [120], the dropout operation leads to the change of the expected output values between the training and the test processes. Thus, the weights need to be adjusted and scaled accordingly during the test process to preserve the expected output values. For example, denote the dropout rate p (probability to drop a neuron) and an arbitrary output entry considering dropout is $z = \mathbf{w}^T \mathbf{x} + b$. The expected value of the entry applying dropout during training is $E(\text{dropout}(z)) = (1-p)z + p \cdot 0$, which is scaled by $(1-p)$. Therefore, during the test process, to preserve the expected value, the weights for computing the output entry z need to be multiplied by $(1-p)$. Equivalently, without adjusting the weights during the test process, the same results can be achieved by normalizing (dividing) the weights by (1-p) during training. Empirically, the dropout rate p is typically taken as about 0.5. The larger the value, the more neurons are possibly dropped during training. Especially for the case of using DL in small-scale dataset, the deep NN containing a large amount of neurons



(a) Standard network (b) After dropout (⊗)

Figure 3.2: Dropout operation in the NN model

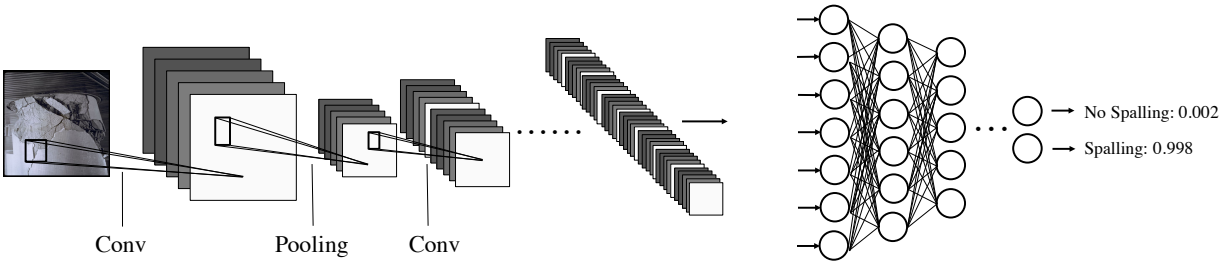


Figure 3.3: Illustration of the Conv and pooling operations in a CNN for vision-based SHM

may suffer from over-fitting due to data deficit. In this case, a heavy dropout rate (say 0.7 as conducted in the experiments of Chapter 5) reduces the trainable parameters in the neurons during training, which may be helpful in alleviating the over-fitting issue.

3.2 Convolutional Neural Network (CNN)

Similar to the NN, a typical CNN is also stacking from multiple layers, but different types of layers are applied besides the fully-connected (fc-)layer used in the NN, e.g., convolutional (Conv) and pooling layers illustrated in Fig. 3.3. As mentioned in Chapter 1, nowadays DL adopting deep CNN is the state-of-the-art approach for vision-based SHM, and several well-known CNN models (architectures), e.g., VGGNet [113], InceptionNet [122], and Residual Network (ResNet) [42], with strong representation ability are widely used in many applications. Before diving into these advanced models, brief mathematical reasoning and basics about how CNN works are presented in this section.

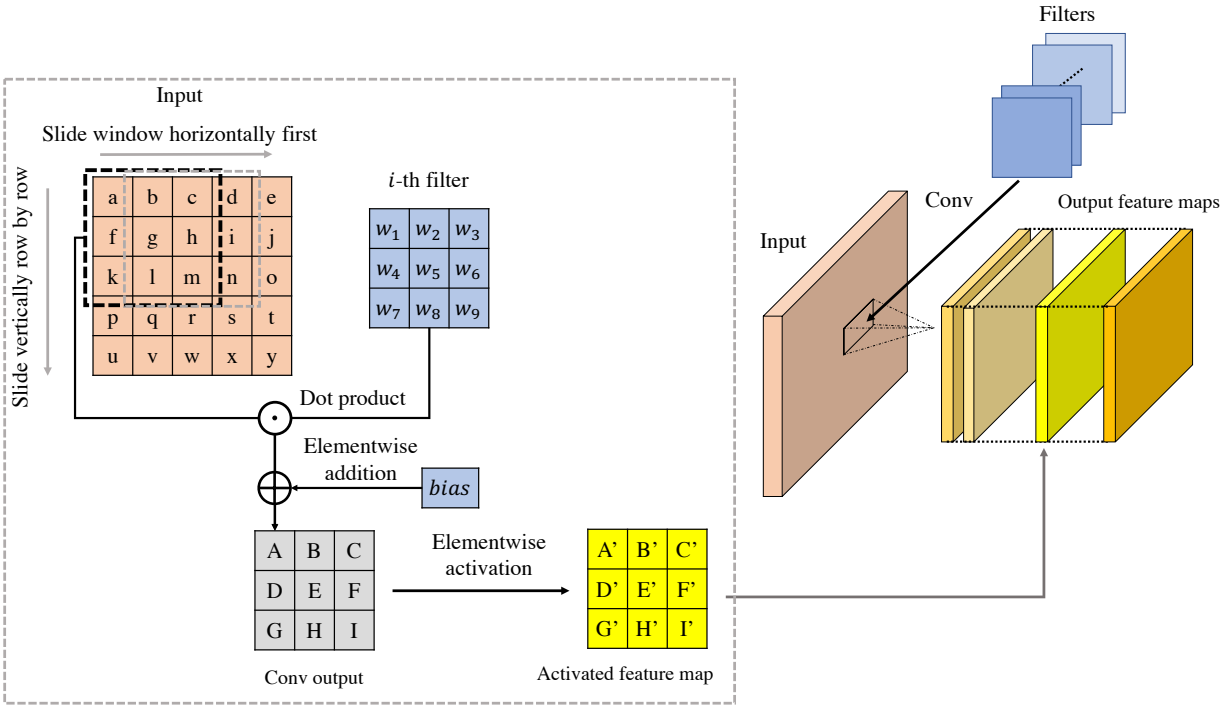


Figure 3.4: Illustration of the Conv operations in a simple example

Convolution

The convolution operation in CNN is performed in the Conv layer through convolving a *filter* (*kernel*) with the previous input in a manner of a sliding window from left to right and from top to bottom. The output of the Conv layer sometimes refers to as the *feature map*, which shares the similar concepts of features in the NN but the feature map usually is a tensor and exists in a more abstract and high-dimensional space. To increase the recognition ability, in each Conv layer, the input is operated with multiple filters, and then the outputs are activated by some nonlinear activation functions (introduced in Section 3.1), where ReLU is the most commonly used one, and accordingly multiple feature maps are generated. Each feature map describes certain information by a matrix in the 2-D space, and multiple features maps produced by the filters are stacked together along the new third axis, which enhances the space from 2-D to 3-D and makes the stacked feature maps represented as a tensor. The mathematical foundation of CNN is introduced through a simple example as illustrated in Fig. 3.4.

Suppose the input is a 5×5 matrix, and it is convolving with the i-th filer, whose size is 3×3. Start from the top left, the first entry of the Conv output "A" is computed by the dot-product of the weight matrix (filter) with the same size of sub-matrix in the top-left corner of the input, and then adding one bias term as follows,

$$\mathrm{A} = \mathrm{a} \cdot \mathrm{w}_1 + \mathrm{b} \cdot \mathrm{w}_2 + \mathrm{c} \cdot \mathrm{w}_3 + \mathrm{f} \cdot \mathrm{w}_4 + \mathrm{g} \cdot \mathrm{w}_5 + \mathrm{h} \cdot \mathrm{w}_6 + \mathrm{k} \cdot \mathrm{w}_7 + \mathrm{l} \cdot \mathrm{w}_8 + \mathrm{m} \cdot \mathrm{w}_9 + \mathrm{bias} \quad (3.12)$$

To increase the nonlinearity, the Conv output is activated by ReLU. Therefore, the first element of the activated feature map "A′" is computed as follows,

$$\mathrm{A}' = \mathrm{ReLU}(\mathrm{A}) = max(0, \mathrm{A}) \quad (3.13)$$

Define the *stride* as the step size of the sliding window, e.g., stride 1 means each time move 1 unit away from the current position. Suppose in the example shown in Fig. 3.4, a stride of 1 is used, then moves the window by 1 unit and repeats the dot-product operation for elements (b, c, d, g, h, i, l, m, and n) and the same weight matrix (filter). Similarly, the element B′ in the active feature map is obtained. Loop and repeat the above-mentioned operations for the entire input matrix with the stride 1 sliding window following the sequence of left to right and top to bottom. Accordingly, a 3×3 activated feature map is generated for the i-th filter.

The stride size S is adjustable, which is a hyper-parameter, and using different stride sizes yield different shape of the feature map. Conventionally, the same stride is used for both horizontal and vertical sliding window. Suppose the shape of the current input is (P, Q), where P and Q represent the size in the horizontal and the vertical directions, respectively, and the size of filter is F (the conventional filter is a square). The output shape (P', Q') is computed as follows,

$$\begin{aligned} P' &= \frac{(P - F)}{S} + 1 \\ Q' &= \frac{(Q - F)}{S} + 1 \end{aligned} \quad (3.14)$$

However, from Eq. 3.14, using $S > 1$ requires the differences between the shape of the input (P, Q) and the filter size F, i.e., $P - F$ and $Q - F$, to be divisible by S, which makes the selection of S limited. Thus, the *padding* technique was developed [58] and introduced next. Note that when Eq. 3.14 is applied to the example in Fig. 3.4, i.e., $P = Q = 5$, $F = 3$, and $S = 1$, one obtains $P' = Q' = 3$.

One Conv layer usually consists of multiple filters, yielding multiple feature maps, where one filter contributes to its corresponding feature map. Suppose there are total of M filters in Fig. 3.4, thus the Conv layer produces M 3×3 feature maps. Therefore, by stacking all the feature maps together, the shape of the final layer output is $(N, 3, 3, M)$, where N is the number of input data. As for a deep CNN, N input data are processed via multiple Conv layers and finally end up with the high-dimensional feature maps with the shape of (N, W, H, M), where W and H present the 2-D dimensional shape of the feature map in the final Conv layer. To obtain the class probability, the feature maps of the final Conv layer are further flatten into the shape of $(N, W \times H \times M)$. Subsequently, the flattened feature

0	0	0	0	0
0	a	b	c	0
0	d	e	f	0
0	g	h	i	0
0	0	0	0	0

Figure 3.5: Example of one unit padding on a 3×3 input

is further connected with neurons in the fc-layers to output the probability values of the different classes of the feature, as shown in Fig. 3.3.

In many cases, e.g., RGB images, the input is usually a tensor, whose fourth dimension represents channel information, where the first three correspond to number of data, input width, and height. Usually, the fourth dimension of the output increases after the Conv operations, which equals to the number of the applied filters (kernels). Accordingly, in the training, the filters are also elevated to the third dimension to be consistent with the fourth dimension of the input. Suppose the current input size is (N, P, Q, C), and the actually shape of the filter to be initialized and trained is (F, F, C). The (F, F, C) filters convolves with the (P, Q, C) input along the first dimension N. Moreover, since all parameters of one Conv layer are weights in the filters and one bias term, it is straightforward to compute the number of parameters N_{para} to be trained in one Conv layer with M $F \times F$ filters, as follows,

$$N_{para} = M \cdot (F \cdot F \cdot C) + 1 \tag{3.15}$$

Padding

From the example shown in Fig. 3.4, it is observed that after one Conv layer, the shape of the feature maps shrinks. Moreover, as mentioned above, using a stride larger than 1 may lead to indivisible numbers, and using a larger size stride also leads to the shape of output diminishing quickly, which causes difficulty in constructing a deep network. Thus, the technique called *zero padding* is introduced, which works by padding a certain number of "0" elements around the raw input. One example of one unit padding on a raw 3×3 input is shown in Fig. 3.5.

Similarly, if padding U units on the surroundings of the input with shape (P, Q) and using the same $F \times F$ filter, the output shape (P', Q') can be computed as follows,

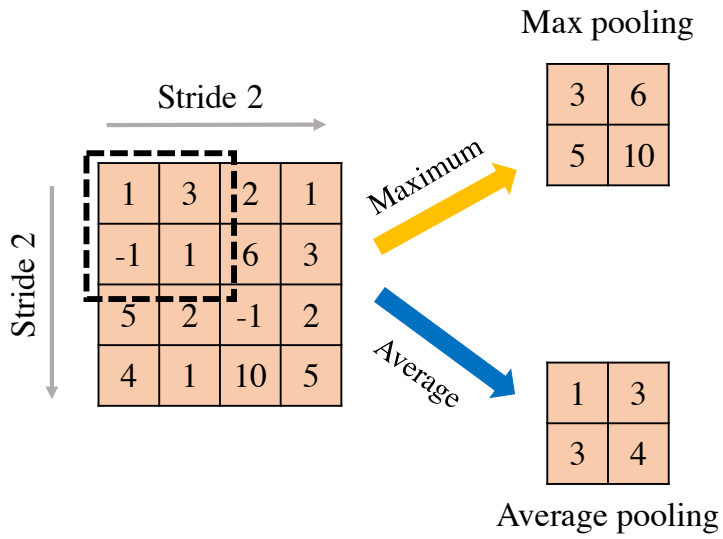


Figure 3.6: Example of max pooling and average pooling

$$P' = \frac{(P + 2U - F)}{S} + 1$$
$$Q' = \frac{(Q + 2U - F)}{S} + 1 \tag{3.16}$$

Mathematically, padding "0" elements does not influence the results of the Conv operations, and also maintains the same shape between the input and the output if using stride 1. In addition, the input becomes divisible if padding an appropriate number of the "0" elements.

Pooling

The pooling layer is another special layer component in the deep CNN. It can reduce the dimensionality of the representation and create an invariance to small shifts and distortions [38]. In a typical deep CNN design, the pooling layers are usually placed after multiple stacked Conv layers.

There are two types of widely used pooing layers, namely *max pooling* and *average pooling*. As reflected by their names, max pooling outputs the maximum element within the sliding window and average pooling outputs the average value of all elements in the window. Usually, the same value is used for both the sliding window and the stride size, which avoids overlapping. The most common combination is to use 2 for both window size and stride size, where using a larger size leads to a more rapid dimensionality reduction. One example with respect to both max pooling and average pooling is shown in Fig. 3.6 using 2×2 window size and stride 2.

It is noted that max pooling has some positive effects in extracting superior invariant features [86], which might help with identifying features, like cracks. Average pooling focuses more on maintaining the integrity of the information while conducting the dimensionality reduction.

Batch normalization

One common issue in DL is that with the increase of the network depth, it becomes more difficult to train and it converges slowly. This issue can be partially attributed to *internal covariate shift* phenomenon. In a deep network, it is known that the output from the previous layer is used as input for the current layer. However, with the increase of depth, the distribution of the input for each layer changes and shifts frequently during the iterations of the training process, and the distributions of the input between layers may differ significantly. It is noted that *covariate shift* usually relates to the input layer, but here internal covariate shift refers to the distribution shift within the network with respect to the input of the internal layers. Therefore, to reduce the internal covariate shift, *batch normalization* (BatchNorm) was proposed [47].

Simply conducting normalization towards the input of each internal layer usually does not work and makes the network learn nothing from the data. Thus, BatchNorm is performed in a slightly different way. Suppose the input batch data from the previous layer is $\{x_1, ..., x_m\}$. During training, after normalizing the input, Eqs 3.17 to 3.19, BatchNorm introduces two trainable parameters, namely the shift parameter β and the scale parameter γ to map the normalized input back to another distribution, which is intuitively thought to be suitable for the network training, e.g., changeable model parameters, decreasing loss, and moderate training speed, Eq. 3.20. In other words, β and γ are automatically learned and adjusted by the SGD algorithm in the training to fit the corresponding distribution. To avoid numerical difficulty through division by zero, a small value ϵ (e.g., 10^{-6}) is added to the denominator in Eq. 3.19. In addition, during the test for prediction, the BatchNorm layer functions differently from the mean μ and standard deviation (Std) σ^2, which are not computed from the test batch but instead taken as fixed empirical values as used in the training.

$$\mu_{BN} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.17}$$

$$\sigma^2_{BN} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{BN})^2 \tag{3.18}$$

$$\hat{x_i} = \frac{x_i - \mu_{BN}}{\sqrt{\sigma^2_{BN} + \epsilon}} \tag{3.19}$$

$$x'_i = \beta + \gamma\hat{x_i} \tag{3.20}$$

In summary, the BatchNorm layer realizes the input normalization, and it also maps the normalized input back to maintain the original data characteristics, where both mapping parameters are automatically learned by the network itself during training. According to [47], BatchNorm helps improve gradient flows in the network, allows using higher learning rates, and reduces the strong dependence on parameter initialization. Besides accelerating the training, BatchNorm somewhat plays the role of regularization, which mitigates the usage of dropout.

3.3 Deep Convolutional Neural Network

Similar to the deep NN, by stacking multiple Conv, pooling, and fc-layers in a certain design, a deep CNN is easily constructed, and its configuration is known as CNN architecture. As mentioned above, several deep CNN architectures have been proposed and validated effectively. This section introduces three popular and widely used deep CNN architectures, namely VGGNet, InceptionNet, and ResNet.

VGGNet

Inherited from the two well-known architectures of LeNet [60] and AlexNet [58], VGGNet uses similar configurations such as multiple *Conv blocks*[1], two fc-layers representing image features, and one fc-layer for classification usage. Compared to previous network architectures, VGGNet goes much deeper, exemplifying the concept of "very deep network" [113], with most Conv layers providing a direction for design and efficient usage of CNN. The most prominent contribution of the VGGNet architecture is to prove that by applying a small filter (kernel) size such as 1×1, or 3×3, and increasing the depth of the network, one can effectively improve the model performance. The pre-trained VGGNet model on ImageNet[2] [24] has very good generalization on other datasets and provides an environment for a transfer learning (TL) study, where the DL classifier can learn from a *source domain* to transfer the learning to another *target domain* with less data.

The input size used in Simonyan & Zisserman [113] is fixed at 224×224, which is consistent with the image size, where the pixel intensity in an image is reduced by the mean RGB channel as the only preprocessing procedure. For the Conv layer, only a small filter size (typically 3×3 or even 1×1) is used for the Conv operations. In order to maintain the size of the feature maps generated by convolution, the stride is taken as 1 with zero padding while performing the convolution. In order to save computational time and pursue better performance of feature translational invariance[3], after several rounds of convolution, max pooling layers are added to the Conv layers. The stride for the max pooling is 2 with a 2×2

[1] In this dissertation, a Conv block is defined as several Conv layers ending with a max pooling layer.

[2] `http://www.image-net.org/`.

[3] Translational invariance means that if inputs are translated, the CNN is still be able to detect the class to which the input belongs.

window size. In the original design, after a series of Conv blocks, three fc-layers (also known as *dense layer* here) follow, where each of the first two layers has 4,096 neurons and the third one has 1,000 neurons matching the 1,000 classes of the detection targets. Nonlinear activation function ReLU is applied in all Conv and fc-layers, except the last fc-layer (also known as *softmax layer*) whose activation function is obviously the softmax function. According to different number of filters in the Conv layer, and different number of Conv layers and fc-layers (namely different depth), VGGNet has several variations, e.g., VGG-16 and VGG-19, where the original VGG-16 network architecture is shown in Fig. 3.7, and the two numbers "16" and "19" stand for the number of weight layers (Conv and Dense layers) in the network. More details can be found in [113].

InceptionNet

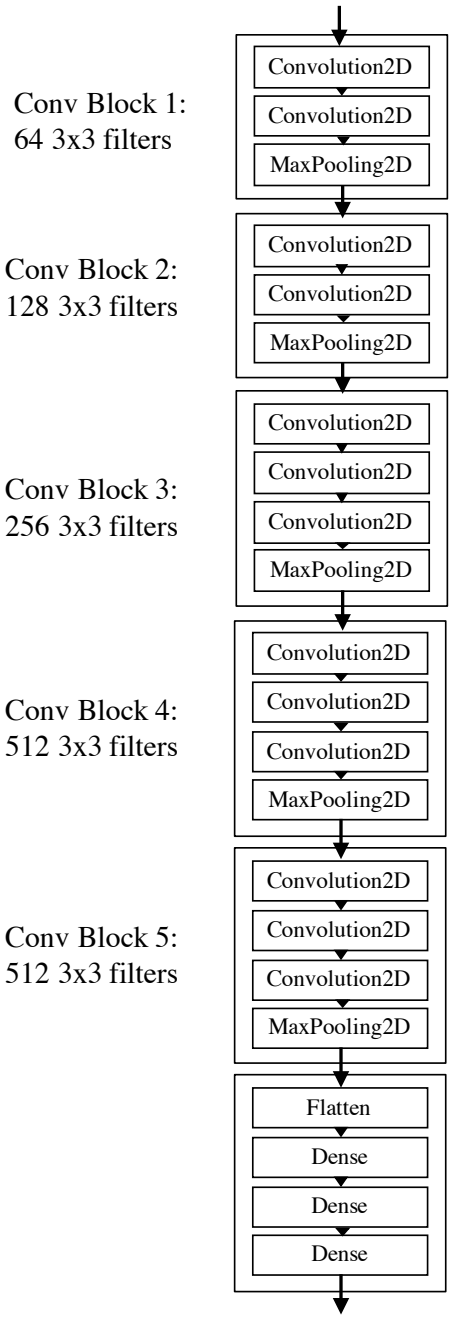
Besides the VGGNet, InceptionNet [122] is another great deep CNN architecture in current DL research and applications. InceptionNet appeared for the first time during the ImageNet Challenge[4] 2014 [24], and it achieved the first place in the classification task with a top-5 6.7% error rate, which out-performed the VGGNet with 7.3% error rate. Afterwards, InceptionNet went through several improvements with different versions, starting from the original InceptionNet V1 [122], to V2 [47], V3 [124], and V4 [123]. Because of its registration name in the ImageNet Challenge 2014, InceptionNet V1 also refers to GoogleNet.

Starting from InceptionNet V1 and compared with the early DL model, e.g., AlexNet [58], InceptionNet V1 significantly decreased the number of network parameters from 6,000 million to only 500 million, which alleviated the high demands on the computational resources. However, the decrease of the number of parameters did not reduce its recognition performance. This is mainly due to several novel changes from the previous designs. These changes are listed below:

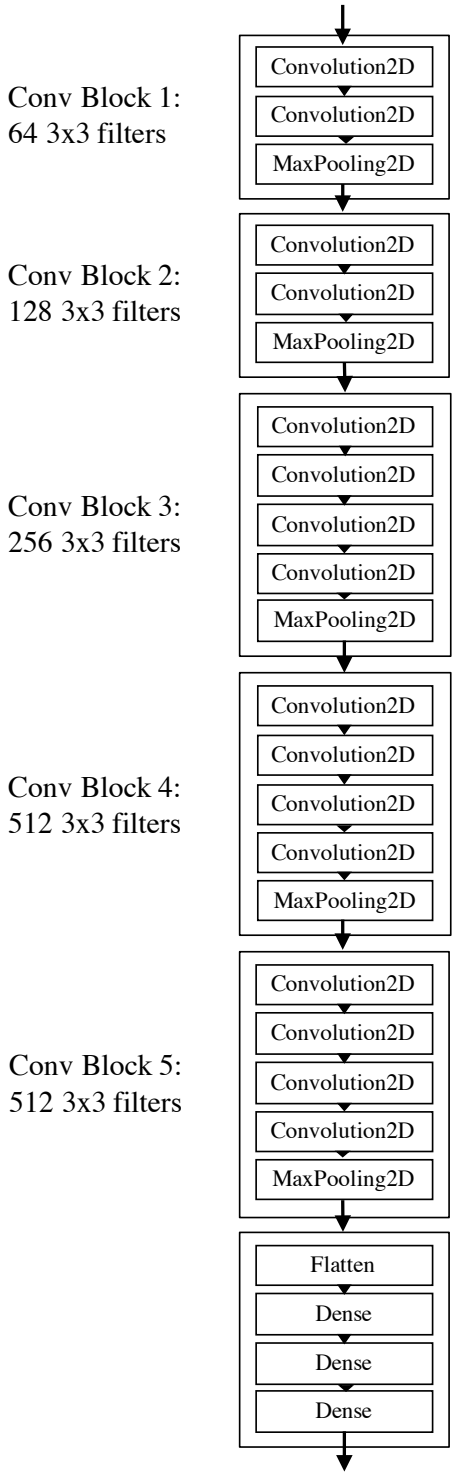
1. Introduce *Inception module* to improve the utilization of parameters.
2. Apply different sizes of filters (kernels) to increase diversity, especially use of multiple small filters to replace larger ones.
3. Add auxiliary classifiers in some middle layers, and add their outputs with small weights into the final output for classification.
4. Remove the last several fc-layers and instead use a global average pooling layer to reduce over-fitting and accelerate training.

Using the Inception module is the core of the network, which is the source of the name of InceptionNet. Essentially, the Inception module is a sub-structure of the network, which is similar to the Conv block used in the VGGNet, and the final deep network is stacked by multiple modules. Unlike in the CNNs where the input signals directly pass through the entire

[4]ImageNet Large Scale Visual Recognition Challenge (ILSVRC).



(a) VGG-16 (13 Conv + 3 Dense)



(b) VGG-19 (16 Conv + 3 Dense)

Figure 3.7: Illustration of the two original VGGNet architectures [113]

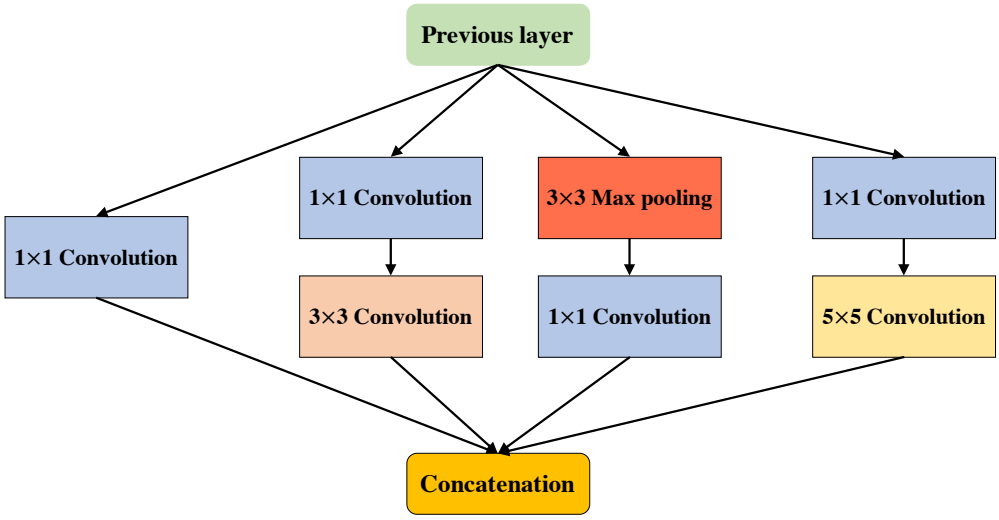


Figure 3.8: Example of one Inception module with four branches

network structure, when the signal from a previous layer passes through the Inception module, it goes through different branches and the outputs from these branches are concatenated and used for the next layer. One example of an Inception module is shown in Fig. 3.8 with 4 branches: the first branch conducts 1×1 Conv operations, the second branch contains two Conv layers to conduct 1×1 and 3×3 Conv operations, etc. It is noted that a 1×1 filter is widely used in these modules, which can transfer information between channels and also help increase or decrease dimensions. There are usually multiple different sizes of filters working on passing the input signal, and the final module output is actually the concatenation of the outputs from multiple different Conv blocks. These characteristics of the Inception module increase both network complexity for recognition and its adaptability to different scales [122].

Based on InceptionNet V1, several modifications were made to further improve its performance. In InceptionNet V2 [47], the 5×5 filter is replaced by using two-layer convolution with two 3×3 filters. In order to reduce the internal covariate shift, BatchNorm is introduced in V2. Moreover, InceptionNet V3 [124] further optimizes the Inception module, and introduces the concept of "Factorization into small convolutions", by changing a relatively large $n \times n$ filter to two-layer convolution with $1 \times n$ and $n \times 1$ filters, which reduces the number of parameters and increases the nonlinearity. Based on V3, InceptionNet V4 [123] combines ResNet [42] (introduced below) within it, which is the state-of-the-art version achieving the lowest top-5 error on the ImageNet. More details can be found in [47, 123, 124].

ResNet

A deep network has more trainable parameters compared to a shallow one, and the increasing complexity might help solve more complicated problems and improve performance. Unfortu-

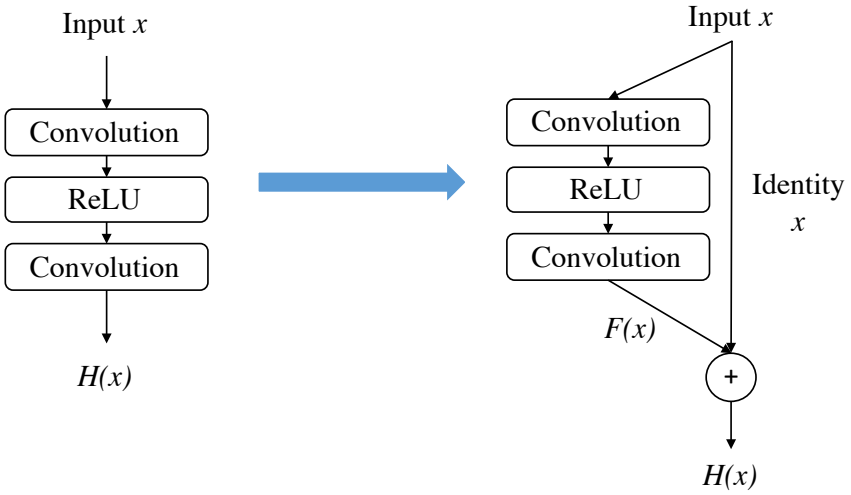
nately, as demonstrated by He et al. [42], vanishing/exploding gradients[5] and degradation[6] issues occur with the increasing of the network depth, which makes it difficult to effectively train.

To address this concern, He et al. [42] developed a new CNN architecture with a shortcut connection named the ResNet. As a reflection of its name, the layers are reformatted with reference to the inputs through the shortcut connection, and the network learns *residual functions* instead of unreferenced functions as is the case with the traditional CNN. These shortcuts act like highways, and the gradients can easily flow back, resulting in faster training and support for stacking more layers. Mathematically, assuming $H(x)$ is any desired mapping function, the traditional way assumes that Conv operations try to fit $H(x)$. On the contrary, in ResNet, the Conv operations are used to fit the residual function $F(x)$, and then mapping can be represented as the sum of functions of input and residual terms, i.e., $H(x) = F(x) + h(x)$. If the additive term $h(x)$ is the original input, i.e., $h(x) = x$, this shortcut connection is considered as an *identity mapping*, Fig. 3.9a, where $\partial H/\partial x = \partial F/\partial x + 1$ improving the gradient vanishing issues and the network learns the residual term $(H(x) - x)$ instead of directly mapping the relationship between the input x and the output $H(x)$. In addition, a more complicated mapping was designed for the shortcut connection using $h(x)$ as the Conv operations, namely *Conv shortcut* or *projection shortcut*, where again the network learns the residual term $(H(x) - h(x))$. It is a usual way to apply 1×1 convolution to match the dimensions between the Conv blocks, where the dimension difference between Conv blocks is mainly due to the applied max pooling layers, which usually down-samples the dimension by 2. Moreover, the dimension difference between layers makes normal identification of the mapping an inapplicable task, Fig. 3.9b. The CNN sub-structures shown in Figs 3.9a & 3.9b are denoted as an identity residual unit and a Conv residual unit, respectively. Moreover, He et al. [43] studied more variants of the shortcut configurations and validated the results with detailed derivations.

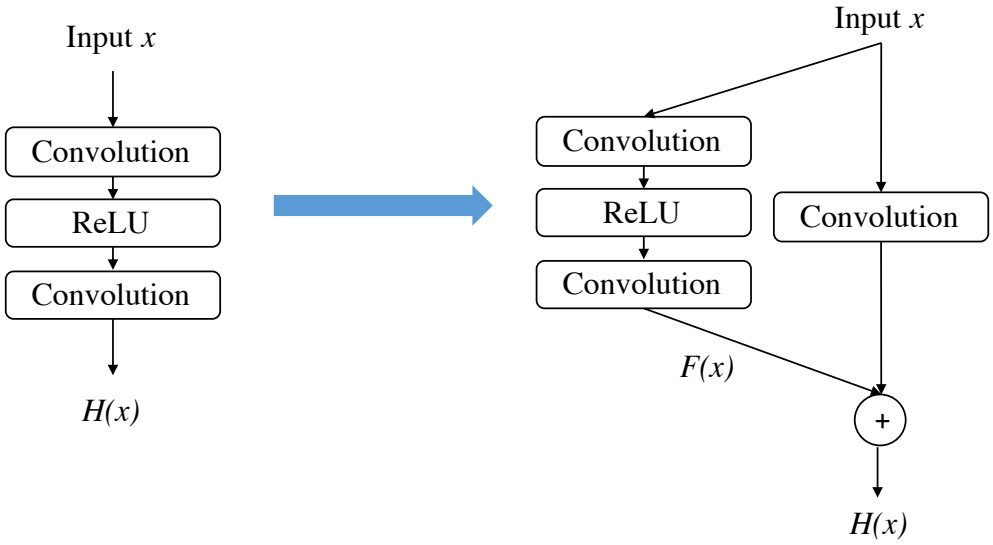
The example of one variant of ResNet, namely ResNet-50, is shown in Fig. 3.10, which consists of stacking multiple identity and Conv residual units, where the first unit in each Conv block is set as Conv unit to match the dimension between layers, and the remaining shortcut units are all identity units. Similar to VGGNet, there are multiple variations of ResNet based on depth, e.g., ResNet-101 and ResNet-152, or the more advanced design, e.g., ResNeXt [128]. Interested readers can find more information in [42, 128].

[5]This is when the values of the gradients diminish to zeros or explode to significantly large numbers, respectively.

[6]By increasing the network depth, the accuracy gets saturated and then degrades rapidly.



(a) Identity mapping



(b) Convolutional shortcut

Figure 3.9: Two types of shortcut connections in ResNet

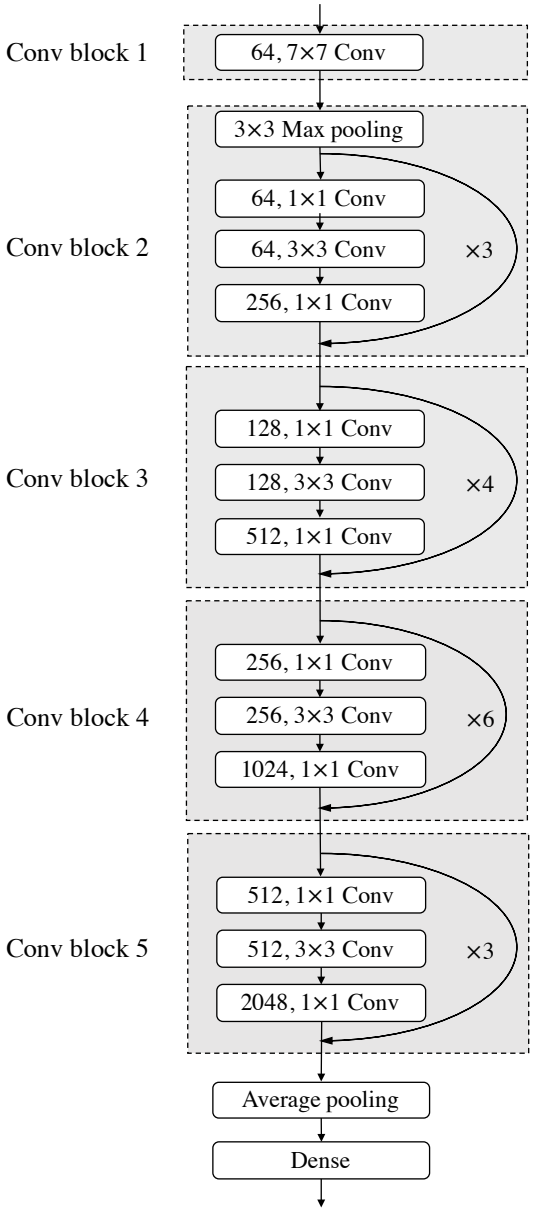


Figure 3.10: Illustration of the ResNet-50 (49 Conv + 1 Dense) network architecture

Chapter 4

PEER Hub ImageNet (ϕ-Net)

Inspired by the establishment of the large-scale high-variety dataset of ImageNet [24], it was proposed to construct a Structural ImageNet, namely PEER[1] Hub ImageNet (ϕ-Net or PHI-Net), which contains images relevant to structural engineering, such as buildings and bridges, with both damaged and undamaged status. Based on past experiences from reconnaissance efforts [55, 65, 106], several structural attributes affecting the structural assessment were considered, i.e., scale of the target object, vision pattern, severity of damage, type of structural component, etc. The constructed ϕ-Net framework and dataset can directly contribute to the development of automatic detection and recognition in vision-based SHM.

4.1 Automatic Detection Framework of the ϕ-Net

The scale of the target object is directly reflected to the distance towards the target. Pre-clustering the images based on such distance, i.e., close, mid-range, and far, related to the scene level (explained later), reduces the problem complexity and subsequent detection can be better performed. Vision patterns and severity of damage can intuitively inform the engineers on how severe the structure is damaged with different criteria based on the scene levels. For images taken from a close or mid-range distance, it is more related to materiel loss and deterioration, i.e., concrete spalling, steel reinforcing bar buckling, etc. On the contrary, for images taken from a far distance, severity is related to the failure status, i.e., complete or partial collapse of a building or a bridge structure. The type of the structural component can provide information for decision-making since the different structural components play different roles in the structural system.

From past experience and domain knowledge, structural vision tasks have certain relationships between each other, i.e., a certain hierarchy exists. Thus, the proposed framework with a hierarchy tree structure is depicted in Fig. 4.1, where the grey boxes represent the detection tasks for the corresponding attributes, white boxes within dashed lines are possible labels of the attributes to be chosen in each task, and ellipses in boxes represent other choices

[1]Pacific Earthquake Engineering Research (PEER) Center.

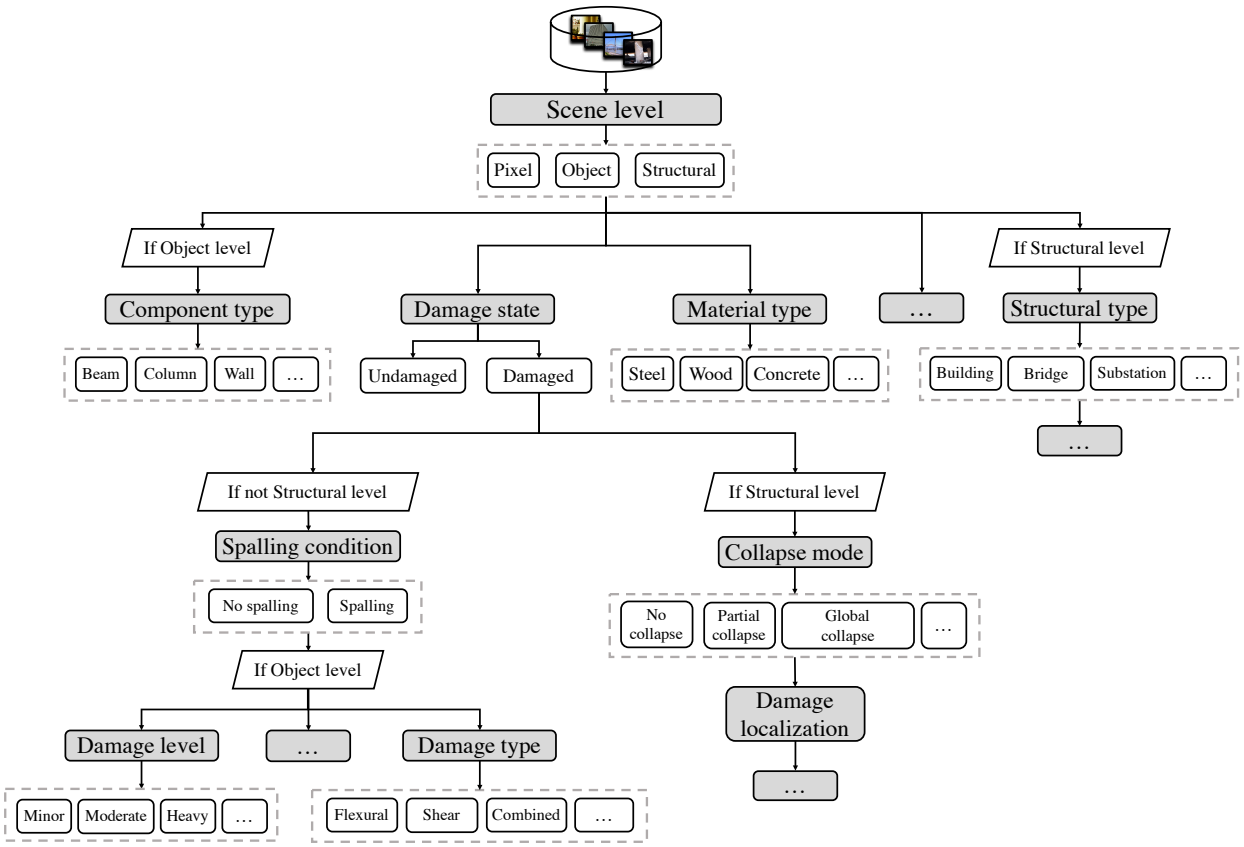


Figure 4.1: Automatic detection framework of the ϕ-Net

or conditions, which represent the possibility for further expansions based on newly collected data or research interests. In the detection procedure, one starts from a root node where recognition tasks are conducted layer by layer and node (gray box) by node until another leaf node. The output label of each node describes a structural attribute. Each structural image may have multiple attributes, e.g., one image can be categorized as pixel level, concrete, damaged state, etc. Thus, in the terminology of CV, this is a multi-attribute classification problem. However, as a pilot study and for a benchmarking purpose, these attributes are treated independently in the current stage.

From the perspective of structural engineering, the structural system is complex. While evaluating the health condition of the structure, information about both global and local details should be considered. For example, a structural elevation image, which belongs to structural level, may reveal the global outside conditions of the structure, e.g., collapse mode and general damage state; then, indoor object level images, collected from key locations of the interior of the structure based on system-level plan, can provide the detailed information for structural components (e.g., column, beam, or foundation), material type (e.g., steel, wood, or concrete), and damage level and type. When only focusing on small regions of the component, pixel level images can be used for local damage severity evaluation, e.g.,

checking spalling. Even though the attribute information of a single image extracted from the ϕ-Net framework is limited, if considered using a large set of images taken from both inside and outside of the target structure, multiple attributes recognized by the DL models are considerably informative. Therefore, assembling information describing certain structural attributes from all collected images can finally lead to a comprehensive view for the structural health condition, which is consistent with post-hazard reconnaissance strategies by human experts.

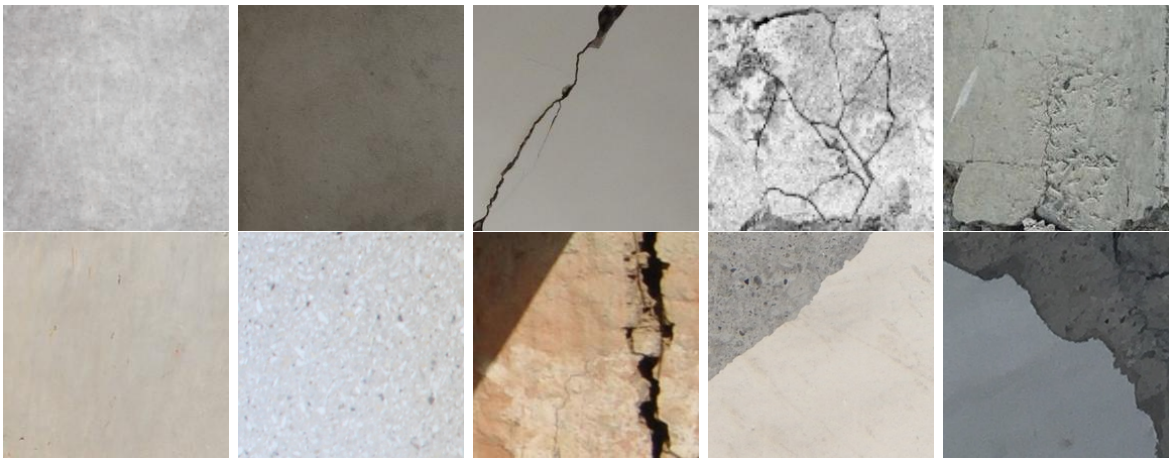
The hierarchy relationship described in Fig. 4.1 is just one reasonable choice among multiple possible combinations. The ϕ-Net framework is generic, and it can be easily adjusted to fit most structural images, i.e., the branches and leaf nodes can be expanded, and node sequences can also be adjusted based on the user's research or application demands. For example, in a later section, several branches are combined and nodes sequence is re-ordered to construct another hierarchy tree structure for a particular scenario.

Benchmark Detection Tasks

In the current ϕ-Net, the following eight classification problems are designed as benchmark tasks based on key attributes: (1) scene level, (2) damage state, (3) spalling condition (material loss), (4) material type, (5) collapse mode, (6) component type, (7) damage level, and (8) damage type. Based on engineering judgment, tasks 1 to 4 are easier than tasks 5 to 8. Moreover, from Fig. 4.1, the hierarchy tree is expandable with more branches and deeper depth, if desired, i.e., boxes with ellipses are possible detection tasks of other attributes for future considerations.

Task 1: Scene level

From Fig. 4.1, the scene level classification acts as an initial task and subsequent tasks are pursued according to its different levels. Because images collected from reconnaissance efforts broadly vary in terms of distance, camera angle, emphasized target, etc., the scene level can help decrease the doubt of the scale issues in the image and reduce the complexity of subsequent tasks. Through filtering out the irrelevant data and dividing images more granularly into different levels, the classifiers for the following tasks can achieve an improved recognition performance. In the current ϕ-Net, the benchmark scene level task is defined with 3-classes: *pixel level* (P), *object level* (O) and *structural level* (S). An image taken from a very close distance or only containing part of the component belongs to P, Fig. 4.2a, an image taken from a mid-range and involving major targets such as single or multiple components belongs to O, Fig. 4.2b, and an image containing most parts of the structure or identifying its outline belongs to S, Fig. 4.2c.



(a) Pixel level



(b) Object level

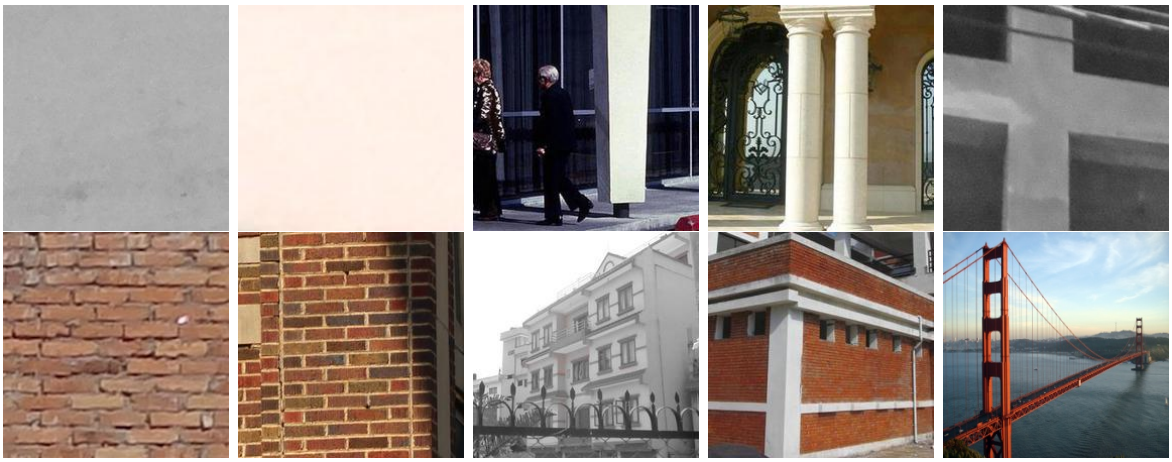


(c) Structural level

Figure 4.2: Sample images used in the scene level task



(a) Damaged state



(b) Undamaged state

Figure 4.3: Sample images used in the damage state task

Task 2: Damage state

In the vision-based SHM, damage state is one of the most important indices for the structural health. In this task, the definition is straightforward where any observable damage pattern on the structural surface implies *damaged state* (D), Fig. 4.3a, otherwise it is *undamaged state* (UD), Fig. 4.3b.

Task 3: Spalling condition (Material loss)

Spalling is usually defined as flakes of material that are broken off of a component or pertains to loss of cover of the component's surface. It often takes place in concrete and masonry structures, which may lead to severe consequences due to reduction in the cross-sectional area and acceleration of corrosion in the reinforcing bars. Spalling can be induced by expansion



(a) Spalling



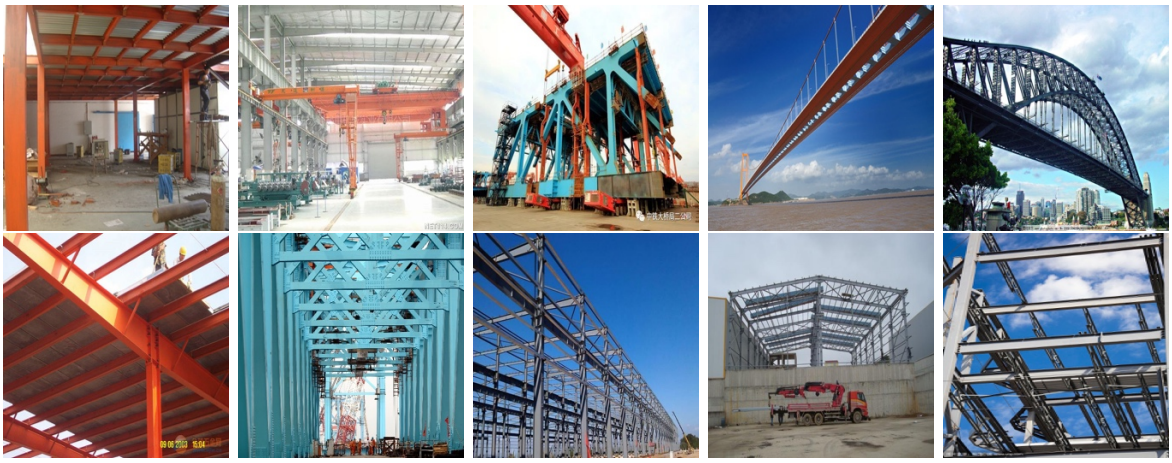
(b) Non-spalling

Figure 4.4: Sample images used in the spalling condition task

forces chemically or mechanically. It is noted that most spalling images here were collected from post-earthquake reconnaissance. In this task, spalling condition is a binary classification task with two classes: *spalling* (SP), Fig. 4.4a, and *non-spalling* (NSP), Fig. 4.4b.

Task 4: Material type

Structural material properties have significant impact on the structural response. Thus, identifying the type of material used for a structural component is important. The commonly used materials are steel, concrete, masonry, and wood. One of the difficulties in this task is that structural components are usually covered with non-structural components, e.g., plaster, making it difficult to accurately identify the material type from surface images. However, several structures and components are made from exposed steel members such as braces and plates and they can be easily recognized by the shape and surface texture. Therefore, in



(a) Steel



(b) Other materials

Figure 4.5: Sample images used in the material type task

this task, simplifications were made to only classify between *steel* (ST), Fig. 4.5a, and *other materials* (OM), Fig. 4.5b.

Task 5: Collapse mode

Due to different photographic distances, camera angles, etc., it is reasonable to evaluate the severity of damage based on different scene levels. In the view of structural level images, which contains the global information of the structure, the collapse mode is defined as *non-collapse* (NC), *partial collapse* (PC), and *global collapse* (GC). NC, Fig. 4.6a, includes undamaged or slightly damaged patterns while the structure is intact. PC, Fig. 4.6b, corresponds to only a part of the structure collapsed while the remaining parts are still intact. GC, Fig. 4.6c, represents catastrophic damage in the structure or evident permanent global excessive deformation.



(a) Non-collapse



(b) Partial collapse



(c) Global collapse

Figure 4.6: Sample images used in the collapse mode task

Task 6: Component type

In the structural system, vertical components, i.e., columns and shear walls provide lateral stiffness to resist lateral forces induced by earthquakes. On the other hand, horizontal components, i.e., beams, diaphragms, and joints transfer the horizontal loads to the vertical components. These components have different importance especially in the seismic design where engineers adopt concepts like "strong column weak beam" and "strong joint weak component" in buildings [82]. Thus, identifying the type of a structural component is informative for SHM. From the hierarchical relationship in the ϕ-Net, component type identification is considered as a subsequent task at the object level with four classes: *beam*, Fig. 4.7a, *column*, Fig. 4.7b, *wall*, Fig. 4.7c, and *others*, Fig. 4.7d. It is challenging to identify whether a wall is a shear wall or an infill wall from the images, even though they have major differences in providing lateral stiffness, thus both are grouped herein into one category. For a component other than beam, column, and wall, e.g., joint, stair, window, or brace, it is treated as others.

Task 7: Damage level

This task refers specifically to the damage level evaluation of images in the pixel and object levels, to complement the collapse mode task at the structural level. It is defined with four classes: *undamaged* (UD), *minor damage* (MiD), *moderate damage* (MoD), and *heavy damage* (HvD). MiD, Fig. 4.8a, implies that there are only small and thin cracks or few small spalling spots on the cover of the structural components. With further development of damage, cracks become wider and spalling areas propagate without failure, indicating MoD, Fig. 4.8b. HvD, Fig. 4.8c, means that the damage area is large and approaching the failure state of the structural component.

Task 8: Damage type

Not restricted to a specific damage representation, e.g., reinforcing bar buckling, concrete spalling, etc., here damage type refers to a complex and abstract semantic vision pattern, defined as *flexural damage* (FLEX), *shear damage* (SHEAR), and *combined damage* (COMB). These damage types have direct implications on the mechanical properties and seismic design. Analogous to the failure types introduced in [82] for seismic loading, definitions are as follows: (1) if most cracks occur in the horizontal or the vertical direction or at the end of a component with vertical or horizontal edges, it is a flexural-type damage, which is more ductile, Fig. 4.9a, (2) if most cracks occur in a diagonal direction, or form an "X" or a "V" pattern, it is a shear-type damage, which is more brittle, Fig. 4.9b, and (3) if the distribution of cracks is irregular, or accompanied with heavy spalling, it is a combined-type damage, Fig. 4.9c.



(a) Beam



(b) Column



(c) Wall



(d) Others

Figure 4.7: Sample images used in the component type task



(a) Minor damage



(b) Moderate damage



(c) Heavy damage

Figure 4.8: Sample images used in the damage level task



(a) Flexural damage



(b) Shear damage



(c) Combined damage

Figure 4.9: Sample images used in the damage type task

4.2 Data Preparation

The procedure to establish the ϕ-Net dataset consists of data collection, preprocessing, labeling, and splitting. These four steps are discussed in the following subsections.

Data collection

To construct a large-scale image dataset, a large number of images is needed, e.g., ImageNet contains over 15 million images. Researchers in structural engineering typically have large numbers of photographs taken from reconnaissance efforts or laboratory experimental investigations, which are good resources for DL training. Moreover, there exist online resources, i.e., databases and search engines storing raw structural images of pre- and post-disaster status but without labels. Besides self-taken and donated images, raw images in the ϕ-Net were collected from NISEE library archive[2], DesignSafe[3], EERI Learning from Earthquake archive[4], Baidu Images[5], and Google Images[6]. Until June 2018, over 100,000 structural images with variant quality were collected, where a large amount was taken after recent earthquakes, e.g., 2017 Mexico City earthquake. As of this writing, multiple worldwide sources contributed to the variety of data in the current ϕ-Net dataset.

Data preprocessing

Since collected images vary broadly with different qualities, image preprocessing was conducted prior to labeling following these four steps:

1. Manually eliminate low-resolution images (lower than 448×448 pixels) and noisy images containing too many irrelevant objects, e.g., vehicles, people, furniture, etc., where the major content of the image is unrelated to damage or other structural attributes from a human perspective.

2. To further increase the dataset size, especially for pixel and object levels, sub-parts, e.g., beam, column, and wall, were cropped from some high-resolution structural images, Fig. 4.10.

3. To avoid significant distortions to the image features due to stretching/rescaling used in training, cropping was applied again to make the aspect ratio of the images roughly less than 2 for the sake of training.

4. To avoid low quality and resolution, cropped images with resolution lower than 224×224 pixels were eliminated.

[2] `https://nisee.berkeley.edu/elibrary`.
[3] `https://www.designsafe-ci.org/`.
[4] `https://www.eeri.org/projects/learning-from-earthquakes-lfe`.
[5] `https://image.baidu.com`.
[6] `https://images.google.com`.

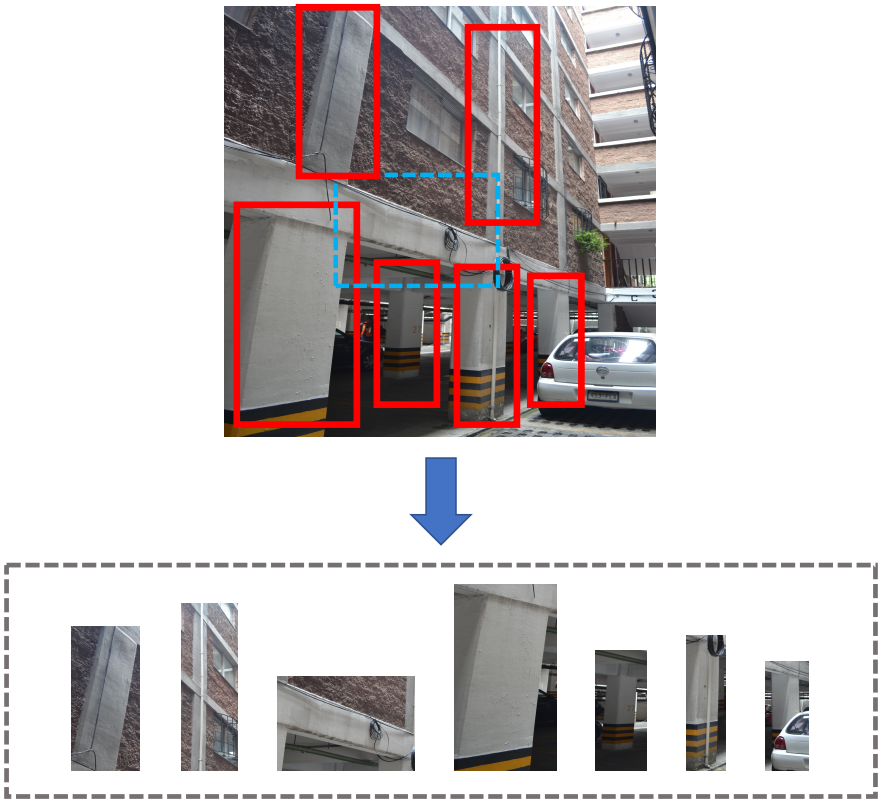


Figure 4.10: Sub-parts are cropped from a high-resolution structural level image

Data labeling

As discussed above, eight benchmark tasks were independently treated. However, in the labeling procedure, the order was followed according to the framework in Fig. 4.1 starting from labeling attribute in the scene level and then progressing towards different tasks. The labeling procedure is crowdsourced via an online labeling tool[7], which provides a user-friendly graphical interface for annotation where users need to only click the button corresponding to the correct label, Fig. 4.11. In this labeling procedure, to avoid bias, majority voting mechanism is adopted where multiple people may label one image and the final label is the one with the majority voting. Moreover, a reference voting mechanism is introduced as shown on the right column of Fig. 4.11, representing the percentages of voting for each choice, which provide opinions from others working on the same image. From the feedback of volunteers, the reference voting mechanism helped in judging some ambiguous images. In some cases, there may exist images unrelated to the proposed attributes herein or the users may not be able to make decisions on certain images. Thus, a "skip" button is added to bypass such images. Once the user completes labeling one task, the selected choices are shown on the bottom right corner, then the user can advance to the next task of the above-mentioned

[7] `https://apps.peer.berkeley.edu/spo`.

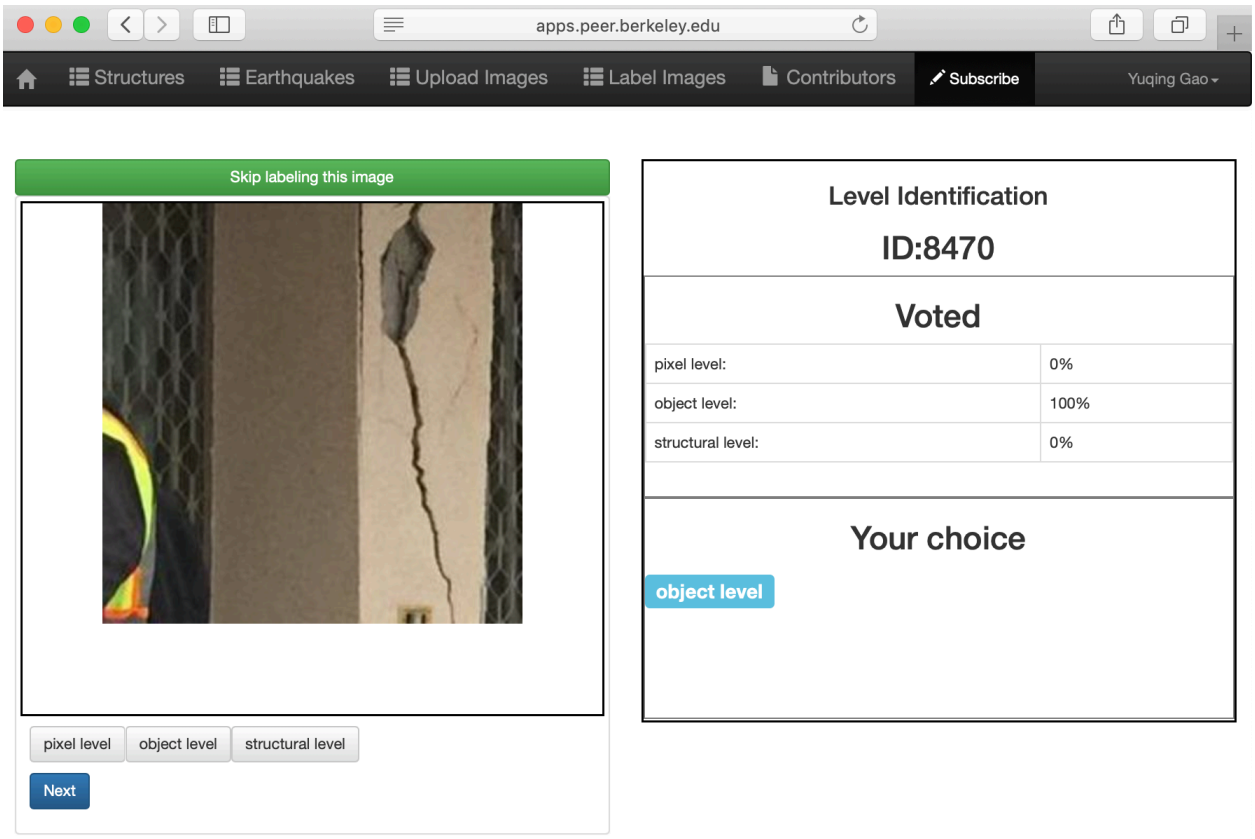


Figure 4.11: Graphical user interface of the online labeling tool

hierarchical relationship by clicking "next".

In the labeling process, 20 volunteers with structural engineering background were recruited. This is in addition to received contributions from worldwide researchers and experts with relevant background knowledge. In order to avoid bias in the current ϕ-Net dataset, each image was labeled with the majority voting among at least three people and was discarded if it has less than three votes. Finally, **36,413** labeled images were obtained and now available in the open-source ϕ-Net database.

Data splitting

As a benchmark dataset for training and validating ML and DL models, splitting raw dataset to deterministic training and test sets is required. However, the ϕ-Net is a multi-attribute dataset where each image contains multiple labels according to relevant attributes, thus simply splitting raw images by a fixed training/test split ratio may lead to different training/test ratios of labels in each attribute (task). This is not the case in common classification benchmark datasets, e.g., MNIST [60] and CIFAR-10 [57]. As an example, in the damage state task, only 13,271 images have valid labels (UD & D), and the remaining 23,142 images

Required: **function** Generate_Training_Index(x) for randomly generating x training indices without replacement; **function** Generate_Test_Index(N, y, *training_index*) to generate y test indices, where $N = x + y$; **function** Compute_Split(task_id, training_index, test_index) to compute the ratio for a specific task.

Define: trial split ratio α, number of total data N_{total}, lower (Bound_L) and upper (Bound_U) bounds of the ratio.

Function Multi_Attribute_Split(α, N_{total}, Bound_L, Bound_U)
    Set done = False
    $N_{training} \leftarrow \alpha \cdot N_{total}$
    $N_{test} \leftarrow (1 - \alpha) \cdot N_{total}$
    while not done **do**
        training_index ← Generate_Training_Index($N_{training}$)
        test_index ← Generate_Test_Index(N_{total}, N_{test}, training_index)
        # Loop over all attributes or tasks
        for task_id in [task_1, task_2, …, task_8] **do**
            ratio ← Compute_Split(task_id, training_index, test_index)
            # Check if violating the boundary assumption
            if ratio < Bound_L or ratio > Bound_U **do**
                # Use recursion
                Multi_Attribute_Split(α, N_{total}, Bound_L, Bound_U)
            end if
        end for
        done ← True
    end while
    return training_index, test_index

Figure 4.12: Multi-attribute split algorithm

are unlabeled for this task. If simply adopting 9:1 training/test ratio across the entire image set, this may lead to the undesirable situation of all images with valid labels to be in the training set with no images with valid labels appearing in the test set. Therefore, a new split algorithm was developed where the training/test ratio was replaced by a range instead of a fixed number for each attribute. Usually, a fixed split ratio for a single attribute dataset is from 4:1 to 9:1, and herein the split ratio range for the multi-attribute ϕ-Net dataset was chosen from 8:1 to 9:1.

Fig. 4.12 shows the pseudo code of the multi-attribute split algorithm. Firstly, according to the number of the whole data ($N_{total} = 36,413$), $\alpha \cdot N_{total} = 32,407$ training indices were randomly sampled without replacement from 1 to 36,413, where the trial split ratio factor $\alpha = 0.89$ was assigned for a better convergence performance after multiple trials, and the remaining 4,006 test indices were generated accordingly. Then training/test ratios were checked for each attribute. If there exist any conflicts where the ratio is out of the range from

8:1 to 9:1, then sampled indices are rejected and run the recursive loop again until it reaches a satisfactory split. Since eight attributes are treated independently, eight task-oriented sub-datasets, one for each attribute with training and test sets, were further separated, as listed in Tables 4.1 to 4.8.

Typically for ML and DL studies, it is suggested to keep a similar label distribution (percentage of each label with respect to all labels) among the training, test, and full datasets. Therefore, the distributions among these eight tasks were examined as shown in Tables 4.9 to 4.16. All results indicated that the ratios among these three datasets to be nearly consistent.

4.3 Data Summary and Extension

In summary, the ϕ-Net dataset is composed of **36,413** effectively labeled images and can be either treated as a one multi-attribute (multi-label) dataset or as consisting of eight independent sub-datasets corresponding to their eight attributes/tasks. The statistics of the labels are shown in Fig. 4.13.

The above-mentioned eight well-labeled datasets with both training and test datasets are open-sourced online[8], and researchers can download the data according to instructions for non-commercial usage. All relevant files are compressed in the form of zip files and share the same data storage architecture containing **X_train.npy** (i.e., training data), **y_train.npy** (i.e., training labels), **X_test.npy** (i.e., test data), and **y_test.npy** (i.e., test labels). All the images are converted to NumPy array and can be easily loaded using several available computer platforms, e.g., Python, MatLab, etc.

The direct usage of the ϕ-Net dataset is to train and validate user's self-developed algorithms and models through a comparison with reported values in this dissertation. Besides the benchmarking purpose, the open-sourced 36,413 images can facilitate future augmentation of self-collected or other open-source datasets mentioned in previous studies with or without labels. For example, the ϕ-Net pixel level concrete images with labels UD and D can be added into SDNET2018 [26] to increase the variety of crack images from multiple structural components and different loading conditions. For the dataset used in [68], the proposed local, component, and system levels are consistent with pixel, object, and structural levels of the ϕ-Net, respectively. Thus, the ϕ-Net images can be combined with this dataset according to these corresponding levels. During the same period of this study, there also exist other SHM studies using a relatively large amount of data, e.g., Yeum et al. [136] conducted spalling and collapse detection problems with thousands of labeled images, but spalling is defined as a localization problem and collapse is treated as a simpler binary detection. For spalling localization, users can search for raw SP images based on SP labels and relabel them with *bounding boxes* to satisfy the label format of localization. For collapse detection, label PC and GC can be merged as label collapse and keep the NC images, which easily satisfies the binary classification format. Similarly, not only for localization, the ϕ-Net dataset can be also extended to other CV tasks, e.g., object detection, segmentation, etc., with extra relabeling

[8] `https://apps.peer.berkeley.edu/phi-net`.

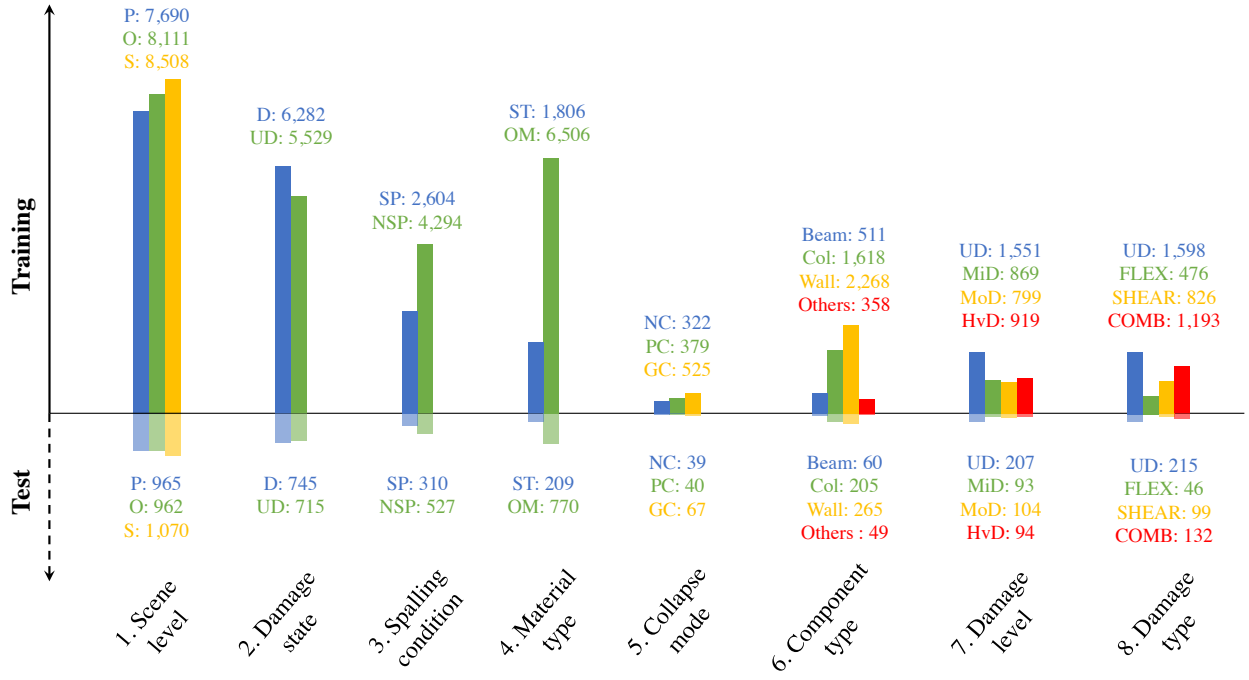


Figure 4.13: Statistics of the labeled data in the ϕ-Net dataset

efforts by the users. In conclusion, complementing the ϕ-Net data increases both number and variety of the data, and it is expected to improve the model accuracy and robustness in similar studies. It is worth noting that this images dataset is not only limited to the scope of DL but can also be beneficial to other data-driven studies using general ML or traditional methods, e.g., [51, 134].

The current ϕ-Net framework and dataset only contain structural surface images and cannot reflect the invisible inner damage. Fusing extra data and information from Lidar [25], 3-D point cloud [16], vibration-based methods using time series signals [103] (refer to Chapters 13 & 14), etc., are worthy of future investigations.

Table 4.1: Number of images in the scene level task

Set	**Pixel level**	**Object level**	**Structural level**	**Total**
Training	7,690	8,111	8,508	24,309
Test	965	962	1,070	2,997
Total	9,073	8,655	9,578	27,306

Table 4.2: Number of images in the damage state task

Set	Damaged state	Undamaged state	Total
Training	6,282	5,529	11,811
Test	745	715	1,460
Total	7,027	6,244	13,271

Table 4.3: Number of images in the spalling condition task

Set	Non-spalling	Spalling	Total
Training	4,294	2,604	6,898
Test	527	310	837
Total	4,821	2,914	7,735

Table 4.4: Number of images in the material type task

Set	Steel	Other materials	Total
Training	1,806	6,506	8,312
Test	209	770	979
Total	2,015	7,276	9,291

Table 4.5: Number of images in the collapse mode task

Set	Non-collapse	Partial collapse	Global collapse	Total
Training	322	379	525	1,226
Test	39	40	67	146
Total	361	419	592	1,372

Table 4.6: Number of images in the component type task

Set	Beam	Column	Wall	Others	Total
Training	511	1,618	2,268	358	4,755
Test	60	205	265	49	579
Total	571	1,823	2,533	407	5,334

Table 4.7: Number of images in the damage level task

Set	Undamaged	Minor	Moderate	Heavy	Total
Training	1,551	869	799	919	4,138
Test	207	93	104	94	498
Total	1,758	962	903	1,013	4,636

Table 4.8: Number of images in the damage type task

Set	Undamaged	Flexural	Shear	Combined	Total
Training	1,598	476	826	1,193	4,093
Test	215	46	99	132	492
Total	1,813	522	925	1,325	4,585

Table 4.9: Label distribution in training, test, and full dataset in the scene level task

Set	Pixel level	Object level	Structural level
Training	31.6%	33.4%	35.0%
Test	32.2%	32.1%	35.7%
Full	31.7%	33.2%	35.1%

Table 4.10: Label distribution in training, test, and full dataset in the damage state task

Set	Damaged state	Undamaged state
Training	53.2%	46.8%
Test	51.0%	49.0%
Full	53.0%	47.0%

Table 4.11: Label distribution in training, test, and full dataset in the spalling condition task

Set	Non-spalling	Spalling
Training	62.2%	37.8%
Test	63.0%	37.0%
Full	62.3%	37.7%

Table 4.12: Label distribution in training, test, and full dataset in the material type task

Set	Steel	Other materials
Training	21.7%	78.3%
Test	21.3%	78.7%
Full	21.7%	78.3%

Table 4.13: Label distribution in training, test, and full dataset in the collapse mode task

Set	Non-collapse	Partial collapse	Global collapse
Training	26.3%	30.9%	42.8%
Test	26.7%	27.4%	45.9%
Full	26.3%	30.5%	43.1%

Table 4.14: Label distribution in training, test, and full dataset in the component type task

Set	Beam	Column	Wall	Others
Training	10.7%	34.0%	47.7%	7.5%
Test	10.4%	35.4%	45.8%	8.5%
Full	10.7%	34.2%	47.5%	7.6%

Table 4.15: Label distribution in training, test, and full dataset in the damage level task

Set	Undamaged	Minor	Moderate	Heavy
Training	37.5%	21.0%	19.3%	22.2%
Test	41.6%	18.7%	20.9%	18.9%
Full	37.9%	20.8%	19.5%	21.9%

Table 4.16: Label distribution in training, test, and full dataset in the damage type task

Set	Undamaged	Flexural	Shear	Combined
Training	39.0%	11.6%	20.2%	29.1%
Test	43.7%	9.3%	20.1%	26.8%
Full	39.5%	11.4%	20.2%	28.9%

4.4 Extensions of the Dataset Hierarchy

The flowchart of the ϕ-Net framework shown in Fig. 4.1 indicates the hierarchical relationship between tasks, and its dependence is based on the engineer's domain knowledge and past experience. One possible hierarchical framework was proposed here, namely hierarchy ϕ-Net, and presented in Fig. 4.14. It combines several branches for efficiency in the computations and introduces conditional nodes (diamonds) for tree-branch selections. The grey nodes in Fig. 4.14 represent detection tasks. All eight benchmark tasks are included; therefore, the ϕ-Net dataset is also suitable for this extended framework. Hierarchy ϕ-Net has a clearer order and stronger correlation between the root and the leaf tasks than the original framework by additional emphasis on the two types of diamond nodes (i.e., scene level and damage state tasks) to make them play the most important roles in the whole framework. The remaining nodes are now connected, and follow a clear sequence. Thus, it is expected that the hierarchy ϕ-Net to be beneficial for future studies. However, the order and selection of nodes (tasks) are not optimal, and more studies to explore and to better utilize the hierarchical relationships are encouraged.

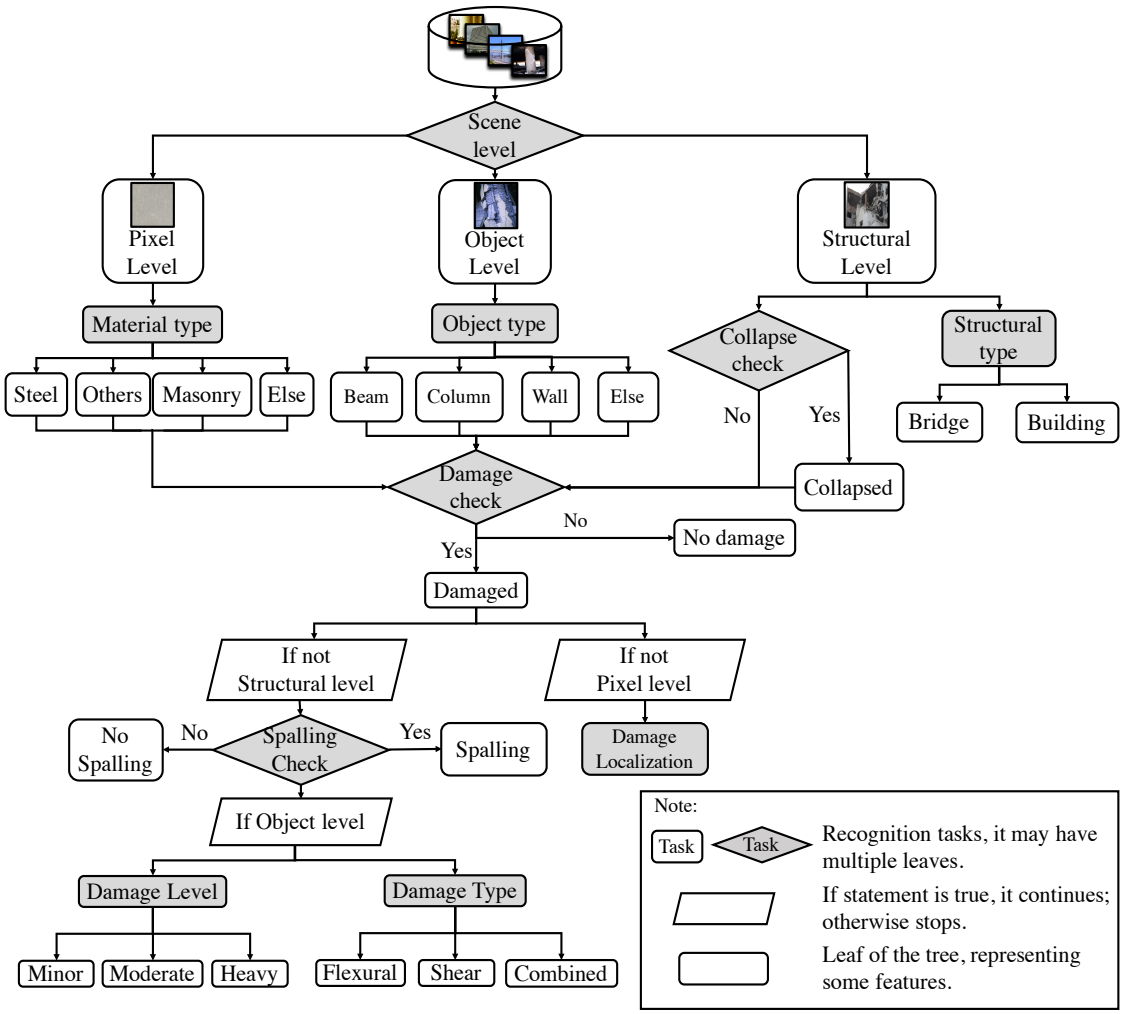


Figure 4.14: Hierarchy of the extended ϕ-Net framework

Shallow depth representation

From Fig. 4.14, the hierarchy ϕ-Net has certain depths representing the complexity of the whole framework and the related problems. Sometimes, if the user only focuses on simpler tasks, e.g., scene level and damage state, instead of using the whole framework, a hierarchical pipeline with shallow depth can be applied, Fig. 4.15. When only using the information until depth-2 layer, it is denoted as depth-2 representation, where the root and leaf tasks are scene level and damage state classifications, respectively. From the nodes in depth-2 layer, the pipeline partitions the damaged and undamaged states into 6 sub-categories, known as leaf nodes as follows: D or UD can be further separated and sorted from (1) the pixel level with damaged state (PD), (2) the object level with damaged state (OD), and (3) the structural level with damaged state (SD) or (4) the pixel level with undamaged state (PUD), (5) the object level with undamaged state (OUD), and (6) the structural level with undamaged state

(SUD).

From the ϕ-net, more information, as prior knowledge, can be added by further expanding the leaf nodes in depth-2 into a deeper layer. Thus, these nodes become intermediate nodes. This is denoted as depth-3 representation if expanded to one more layer. The leaf nodes in the depth-3 layer represent sub-categories in more fine classification tasks, e.g., component type, spalling condition, collapse check, etc. Accordingly, depth-3 representation has lower variations within the sub-classes than depth-2, but it may lose information by excluding some sub-classes leading to possible less diversity. Moreover, for the CV problem, leaf nodes in depth-3 are still thought of as being complex due to different lighting conditions, shapes, textures, surrounding objects, damage patterns, etc. Thus, dealing with images in the subsets of the leaf nodes is still challenging compared to several well-studied CV applications, e.g., handwriting digits [60] and face recognition problems [97].

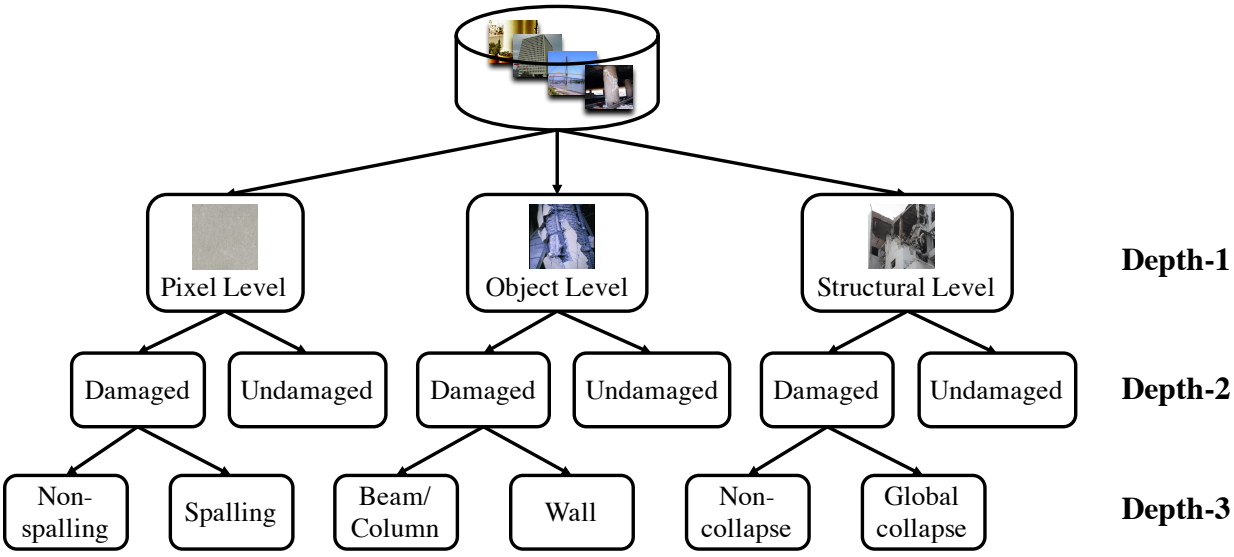


Figure 4.15: Shallow depth representation of the ϕ-Net

Chapter 5

Transfer Learning Using a Small Structural ImageNet

ML and DL can achieve promising or even surprising results through training on a big dataset, but the performance may degrade on a smaller-scale dataset. However, the terms "big" and "small" of the scale of the dataset are ambiguous and subjective, because there is no explicit definition or specific threshold value to well separate them. It is also inappropriate and impossible to infer the results simply by examining the scale of the collected dataset without validation experiments. Similar issue occurs in studying DL performance for the eight ϕ-Net benchmark vision tasks, discussed in Chapter 4, where the numbers of labeled data in the eight ϕ-Net sub-datasets vary from 1,372 to 27,306 images. Compared to traditional studies in vision-based SHM, these datasets are thought to be "big"; however, when compared to common CV benchmark dataset, e.g., ImageNet [24], MNIST [60], and CIFAR-10 [57], these eight datasets are too "small". Therefore, to validate the performance of using DL methods in these datasets, comprehensive benchmarking experiments are required.

Conducting comprehensive numerical experiments on the eight sub-datasets of the ϕ-Net is highly resource-consuming. Before doing so, a preliminary study was performed firstly on a smaller-scale dataset for validating the effectiveness and applicability of DL methods in vision-based SHM, where data were manually and carefully selected from the original ϕ-Net sub-datasets. In addition, to alleviate the training and over-fitting issues while dealing with such a small-scale dataset, one common DL strategy, namely transfer learning (TL), was introduced and applied to the deep CNN models. Besides investigating the performance of the DL models on small-scale datasets, two strategies/patterns of TL, namely feature extractor and fine-tuning, were also explored and discussed.

5.1 Transfer Learning (TL)

In some engineering fields, ML has already achieved a great success in recent years, but it heavily relies on large amounts of data especially labeled ones. In real applications, sometimes

useful "*big data*" is very rare or expensive and the worst case is that only few data can be collected. In order to mitigate the great dependency of ML accuracy on available data size and to maximize usage of existing data, TL became a very effective tool. The general definition of TL [93] is that: Given a *source domain* and its learning task, a *target domain* and its target task, the objective of the TL is to improve the prediction function in learning the target task in the target domain using the knowledge from the source domain with the source target. According to different conditions between the source and target domains and tasks, TL can be classified into three sub-classes: inductive, transductive, and unsupervised [93]. The eight benchmark detection tasks in the ϕ-Net belong to the *inductive learning*, which requires a few labeled data in the target domain. Moreover, there are four common approaches that can be applied according to the requirements of the knowledge transfer from the source domain to the target domain: (1) instance transfer, (2) feature representation transfer, (3) parameter transfer, and (4) relational knowledge transfer [93].

In recent years, significant efforts took place in applying TL to visual classification problems, especially in deep CNN. Usually one major issue in training deep CNN is that with the increasing depth of the network, the model becomes very complex requiring excessive time and computational resources. In addition, with the deficiency of training data, over-fitting and difficulties in training such as training accuracy saturation emerge as other problematic issues. Thus, as discussed above, TL can improve learning which model to use and greatly improve the efficiency of the training process.

Compared with general TL approaches, the details of using TL in CNN are somewhat different. In the CNN, the parameters in the *shallow layers* represent low-level features, such as color, texture, and edges of images, while parameters in the *deep layers* attempt to capture more complicated and abstract high-level features [140]. Therefore, the major objective of TL in CNN is to make use of parameters in a well-trained model from the dataset in the source domain to improve the training the dataset in the target domain. If the two datasets are similar, some low-level features of the CNN model are proposed to be similar, which can be shared, and the high-level features can be tuned by TL. Thus, usually in CNN, TL plays the roles of feature representation transfer and parameter transfer [107].

The original training dataset for pre-trained models, e.g., VGGNet, InceptionNet and ResNet, is the ImageNet [24], which includes thousands of images related to buildings, bridges, pillars, walls, etc., belonging to the civil (e.g., structural) engineering field. Therefore, for benchmark detection tasks in the domain of structural engineering, the source and target domains are assumed to be related. Moreover, from the perspective of task objectives, for example, the component type classification between beam, column, wall, and others is a similar but easier task than that of the ImageNet classification problem, since it reduces the 1,000 classes to only 4 classes. According to the domain knowledge of structural engineering and the characteristics of the parameter transfer, low-level features in CNN might be useful for adapting to new tasks, such as texture might be beneficial to spalling condition identification, and pixel change in directions might provide information for damage type though understanding the cracking morphology. Moreover, mid- to high-level features can be learned by retraining some parts of the network. This is illustrated in Fig. 5.1 where two

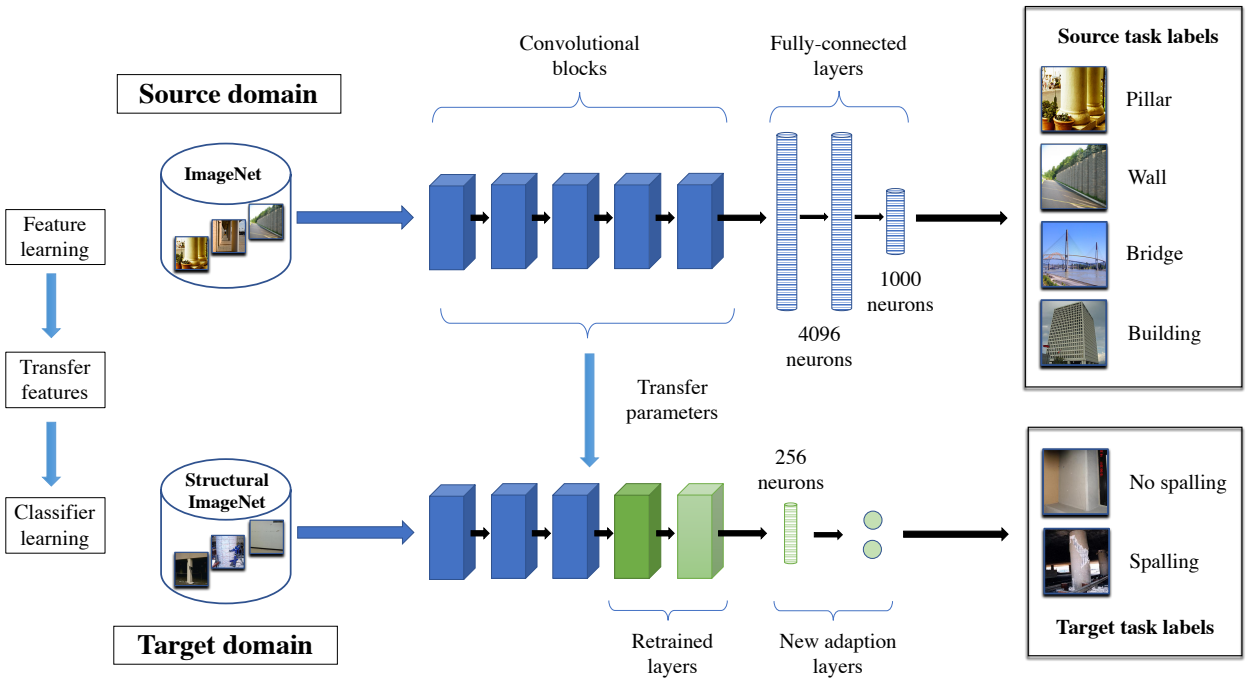
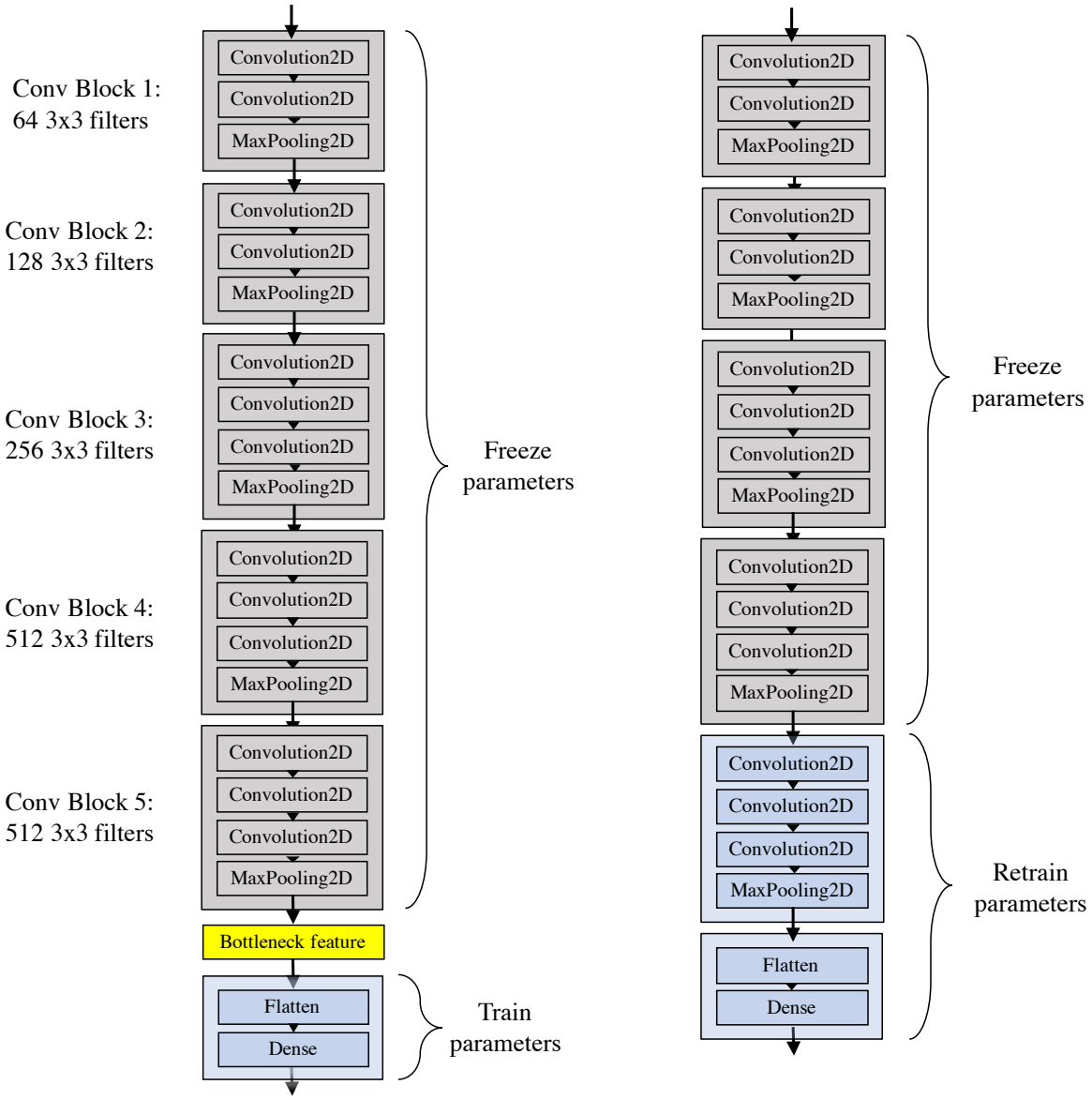


Figure 5.1: Transfer learning flow chart used in this dissertation

Conv blocks are retrained. In addition, the mechanism of TL can be explained in a feature transfer manner. As mentioned above, CNN's parameters represent some features, training on the source domain plays a role as feature learning, and retraining on the target domain is actually transferring some features from the source domain to the target domain and thus learning the classifier for the target domain. During implementations, these pre-trained parameters can relax the procedure of parameter initialization, which might be helpful in dealing with training saturation issues occurring in small datasets.

In general, there are two different strategies/patterns while conducting TL in deep CNN: feature extractor and fine-tuning. Considering as an example the configuration and procedure for normal training in VGGNet [113], feature extractor, and fine-tuning are illustrated in Fig. 5.2. Moreover, since we focus on a small dataset, while implementing fine-tuning, data augmentation (DA[1]) tricks, such as zooming in/out, translation, reflection, rotation, and changing colors, are used to avoid over-fitting.

[1]DA of images data has two types: *online* where the CNN model is trained with processed batch of images via several forms of transformation (e.g., rotation and translation) at each training iteration, and *offline* where new images are generated from transformation of the raw images and added into the original training set before the training process starts, and then the CNN model is trained using this fixed and augmented image set.



(a) Feature extractor (b) Fine-tuning

Figure 5.2: Transfer learning configuration of VGG-16

Feature extractor

Feature extractor (sometimes referred to in literature as "off-the-shelf CNN" [109] or "bottleneck feature for CNN" [116]) performs feature extraction. In this case, the Conv operations are only performed once in the feed-forward procedure. The outputs of the last layer before the fc-layers are taken as the bottleneck feature of the training data, then multiple classifiers can be applied for the classification problem at hand, refer to example of feature extractor in VGGNet, Fig. 5.2a.

The advantage of implementing the feature extractor in TL is that it greatly decreases the training time and number of epochs due to already extracted features. Instead of training a large number of Conv layers for multiple epochs, only features are fed into a shallow NN stacked by the fc-layers or even linear classifiers such as SVM, which are faster to train. Although the Conv operation is computationally expensive and still needs a long time to extract the features by passing the input (e.g., images) through the CNN layers with pre-trained filter parameters and weights, this procedure is only performed once. Thus, all feature extractor procedures require much less computing time than other methods not involving TL. However, one shortcoming of the use of the feature extractor is that *online* DA would not work for this procedure, since the feature is only extracted once, but *offline* DA can be applied to increase the number of datasets, which may lead to a large increase of the data storage demands and the extraction time of the feature.

Fine-tuning

Compared with the feature extractor, fine-tuning retrains some parts of the CNN. Empirically, fine-tuning retrains the Conv blocks instead of just retraining fewer Conv layers, refer to Fig. 5.2b. As mentioned before, Conv blocks in the beginning represent low-level features, which might be similar for both source and target domains, and the objective of fine-tuning the last several Conv blocks is to adjust the mid- to high-level features to the target domain.

In fine-tuning, DA tricks can be applied in the training to avoid the over-fitting problem. The trade-off is that the new augmented images go through a forward propagation at each training epoch, which is time-consuming due to expensive convolution computations. However, the parameters in some parts of the layers are fixed where their backward calculations are skipped and only parameters in the unfixed layers are adjusted by the gradient descent algorithm in the backpropagation. Thus, the use of fine-tuning is still less time-consuming than the regular CNN training from scratch. Moreover, Graphic Processing Unit (GPU)-based parallel computing can also alleviate the excessive computational time demand, which has been shown to be efficient in many civil engineering applications [1, 2].

5.2 Experimental Objectives

Common sense implies that a high-resolution image contains more information with large size of pixel matrices, but the large image occupies more memory, and requires large *network*

input size[2], which is limited by the computer hardware. On the contrary, the small image costs less computation resources but it might lose some information and possibly produce misleading results. In the case of vision-based SHM, low-resolution images might make it difficult to identify minor damages, which can lead to unreliable results for damage evaluation. Thus, the process of vision-based SHM is usually a trade-off between computational efficiency and recognition accuracy. Therefore, the selection of the network input size is part of the network design. The common input size of the input layer used in most popular deep CNN architectures, e.g., VGG-16, VGG-19 [113], and ResNet-50 [42], is 224×224. During training, if the images are either larger or smaller than 224×224, they should be rescaled or cropped to exactly fit the 224×224 size. Thus, it is expected to investigate what is an appropriate network input size for deep CNN in vision-based SHM problems. In order to obtain such an appropriate input size, four network input/image size[3] categories were designed and rescaled for computer experiments, namely *VGG size* (the size used in the prototype VGGNet), *large size*, *medium size*, and *tiny size*, refer to Table 5.1 for details of these empirical choices.

Table 5.1: Details of the network input size experiment

Category	**Large**	**Medium**	**VGG**	**Tiny**
Size	448 ×448	256 ×256	224 ×224	128 ×128

While conducting TL with fine-tuning strategy, the ratio of the retrainable parts (or the fixed parts) of the deep CNN is another hyper-parameter. Even though in many CV applications, the deep CNNs are retrained thoroughly by fine-tuning parameters in all Conv layers per epoch, sometimes this strategy is inapplicable due to limited computational resources and because it may lead to severe over-fitting issues due to limited training data. Thus, only retraining partial network reduces the number of trainable parameters and decreases the model complexity, which may alleviate the over-fitting. However, only retraining a small ratio of the network may also lead to under-fitting issues, since some basic features in the early layers may be different between the two domains which may also need to be fine-tuned similar to the later layers. Therefore, the effect of the ratio of the network part to be retrained is worth studying.

There are many network architectures that can be applied in DL. Moreover, different networks may be more suitable for different problems/tasks. Thus, the selection of the utilized network architecture is another important hyper-parameter to be studied.

In summary, the computer experiments in this chapter include three types. These are as follows: (1) network input size experiment, (2) retrain ratio experiment, and (3) network comparison experiment.

[2]The network input size, (N, W, H, C), is the internal property of the network itself, where N is the number of data (images) fed to the network, and W, H, and C correspond to the designated image width, height, and channels, respectively.

[3]For simplicity, in this part of the study, network input size is consistent with the used image size.

5.3 Experimental Setup

In this section, firstly, the preparation procedure of a simplified small-scale structural image dataset is introduced. The source of this dataset is ϕ-Net, refer to Chapter 4. Then, utilized networks together with the TL training configurations for the three experiments are presented in detail.

For simplicity, in the network input size and retrain ratio experiments, only VGG-type networks (some variants of the original VGGNet) were implemented to obtain the preliminary results. It is noted that based on multiple trials, a similar performance or trend through using feature extractor and fine-tuning with respect to the input size was observed. Therefore, for more computational efficiency and for brevity of this part of the study, in the network input size experiment, only feature extractor was applied. In the retrain ratio and network comparison experiments, the input size was fixed as the default of 224×224. Moreover, in the network comparison experiment, the VGG-type network was further compared with both ResNet-50 and ResNet-101.

The implementation of the TL experiments was conducted on the TensorFlow platform[4] and performed on Alienware 15R3 with single GPU (CPU: Inter(R) Core i7-7700HQ @2.80GHz, RAM: 16.0 GB & GPU: Nvidia Geforce GTX 1060).

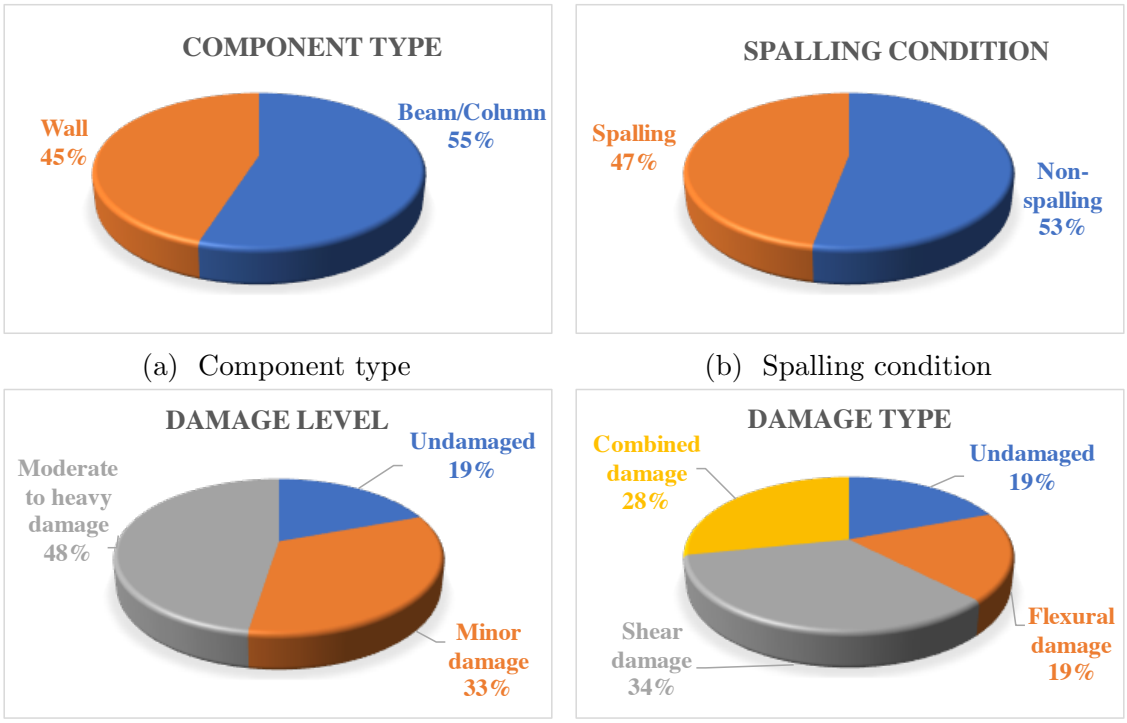
ϕ-Net-2000 dataset

In the above-mentioned three experiments, instead of investigating all eight tasks in the ϕ-Net, the scope of the study is only limited to four classification tasks (component type, spalling condition, damage level, and damage type). Moreover, simplifications are introduced in the class labels for the component type and the damage level tasks. These modifications are listed as follows:

- In the component type task, the original labels "beam" and "column" are combined to a more broad class "beam/column", and the other component types ("others") in the ϕ-Net are not considered here. Thus, it becomes a binary classification between two classes: *beam/column* and *wall*.
- In the damage level task, the original labels "moderate damage" and "heavy damage" are combined to "moderate to heavy damage". It is now a three-class classification: *undamaged state*, *minor damage*, and *moderate to heavy damage*.

In these experiments with a small-scale dataset, 2,000 images from the ϕ-Net dataset are selected, pre-processed, and relabeled manually based on the above modifications, which formed the required small-scale dataset (ϕ-Net-2000) for the TL experiments. It is noted that all 2,000 images have four labels with respect to the considered four detection tasks, and the label statistics for each task are shown in Fig. 5.3.

[4] `https://www.tensorflow.org/`.



(a) Component type (b) Spalling condition

(c) Damage level (d) Damage type

Figure 5.3: Label ratios in the ϕ-Net-2000 dataset

For the two binary tasks, the label ratios are 55/45% and 53/47%, judged to be balanced datasets. For the other two multi-class tasks, since there are no dominating labels but only fewer data for the undamaged cases (no crack and no damage) and considering the fact that a multi-class labeled data is expensive, the shown distributions in Fig. 5.3 are thought to be acceptable. Moreover, although it may not be acceptable to detect an undamaged structure as damaged, it is totally unacceptable and dangerous to do the converse, that is, detect a damaged structure as undamaged. Therefore, using such ratios (with less percentage of undamaged labels) for multi-class detection tasks is reasonable for a practical engineering perspective.

Network input size experiment

Network configuration

In the network input size experiment, the used CNN architectures are variants of VGG-16 (Model D) in [113]. As introduced above, VGG-16 has a fixed input size 224×224, which only receives the image with 224×224 image size. To investigate the influence of different network input sizes, besides VGG size, the network input size of the prototype VGG-16 was modified to accommodate three additional input sizes (large, medium, and tiny), Table 5.1.

For simplicity, while conducting experiments for these four different network input sizes, the images were directly rescaled to match the network input size respectively without additional operations, e.g., cropping. For example of using tiny size in the network input layer, all images were rescaled from their raw image size to a fixed 128×128 size to match the network input.

From the perspective of CV, structural images including damage patterns are much different from natural objects in ImageNet [24]. In TL, usually compensation is made due to different image statistics between the source and the target domains, such as type of objects, typical viewpoints, etc. [92]. As a result, several modifications were made to remove the original fc-layers and instead use adaptation fc-layers (denoted as the *adaptation layer*) in the target domain including the following: (1) different neurons and different numbers of fc-layers were applied after the Conv blocks, and (2) the shape of the softmax layer (the last fc-layer) differed depending on the number of classes to be identified. The dimension of the adaptation layers usually depends on empirical experience and cross-validation. Compared with training using the dataset of the prototype VGG-16, i.e., with over 15 million images, the ϕ-Net-2000 dataset, which was designed for the TL experiments, only has 2,000 well-defined and labeled images. Thus, compromising in the absence of "big data" and to avoid severe over-fitting issues, fewer numbers of neurons and fewer fc-layers were placed after the last pooling layer. Preliminary computations with 256, 1,024, and 4,096 neurons in the first adaptation layer resulted in insignificant differences. However, over-fitting took place in the cases with large numbers of neurons. Thus, considering the cost of computations, an adaptation layer with 256 neurons was a reasonable choice. In the final design, two adaptation fc-layers were used where the first has 256 neurons and the second is a softmax layer, which was changed according to the number of classes to be identified in the classification tasks. Compared with prototype VGG-16, this new configuration makes the network more computationally efficient and compatible with the TL due to data deficiency. The detailed configurations including filter sizes and output shape for each layer of these VGG-type networks (variants of the VGG-16) adapted for the ϕ-Net-2000 dataset are listed in Table 5.2, where N denotes the number of data input to the network, i.e., 2,000 colored images with an initial input depth of 3 for the RGB channels. The output designations 2/3/4 correspond to the different classes to be classified in the adopted four detection tasks, i.e., 2 for the tasks of Figs 5.3a & 5.3b, 3 for the task of Fig. 5.3c, and 4 for the task of Fig. 5.3d.

As for computational efficiency, even though similar CNN architectures were used in the four tasks for the feature extraction, the number of neurons in the fc-layers was different due to different network input sizes and different number of classes in each task. Because of the dimension shrinkage of the fc-layers with fewer classes, the difference of the parameters numbers for different tasks (more precisely, number of classes to be classified) does not significantly affect the total number of parameters, but the input size does. It is easy to understand that in the feature extractor where the Conv blocks are used for feature extraction, the larger the input is, the more the features to be extracted leading to a large increase of the number of neurons in the first fc-layer. Quantitatively, the number of parameters in the large size is nearly 3 times that in the VGG size and 23 times that in the tiny size. Thus, as

Table 5.2: VGG-type network configuration

Conv net architecture			Output shape of models with different input sizes			
Block	**Layer (type)**	**Filter size (#)**	**VGG size**	**Large size**	**Medium size**	**Tiny size**
Input	Input/Image	-	$(N, 224, 224, 3)$	$(N, 448, 448, 3)$	$(N, 256, 256, 3)$	$(N, 128, 128, 3)$
Conv Block 1	Convolutional	3×3 (64)	$(N, 224, 224, 64)$	$(N, 448, 448, 64)$	$(N, 256, 256, 64)$	$(N, 128, 128, 64)$
	Convolutional	3×3 (64)	$(N, 224, 224, 64)$	$(N, 448, 448, 64)$	$(N, 256, 256, 64)$	$(N, 128, 128, 64)$
	Max Pooling	-	$(N, 112, 112, 64)$	$(N, 224, 224, 64)$	$(N, 128, 128, 64)$	$(N, 64, 64, 64)$
Conv Block 2	Convolutional	3×3 (128)	$(N, 112, 112, 128)$	$(N, 224, 224, 128)$	$(N, 128, 128, 128)$	$(N, 64, 64, 128)$
	Convolutional	3×3 (128)	$(N, 112, 112, 128)$	$(N, 224, 224, 128)$	$(N, 128, 128, 128)$	$(N, 64, 64, 128)$
	Max Pooling	-	$(N, 56, 56, 128)$	$(N, 128, 128, 128)$	$(N, 64, 64, 128)$	$(N, 32, 32, 128)$
Conv Block 3	Convolutional	3×3 (256)	$(N, 56, 56, 256)$	$(N, 128, 128, 256)$	$(N, 64, 64, 256)$	$(N, 32, 32, 256)$
	Convolutional	3×3 (256)	$(N, 56, 56, 256)$	$(N, 128, 128, 256)$	$(N, 64, 64, 256)$	$(N, 32, 32, 256)$
	Convolutional	3×3 (256)	$(N, 56, 56, 256)$	$(N, 128, 128, 256)$	$(N, 64, 64, 256)$	$(N, 32, 32, 256)$
	Max Pooling	-	$(N, 28, 28, 256)$	$(N, 56, 56, 256)$	$(N, 32, 32, 256)$	$(N, 16, 16, 256)$
Conv Block 4	Convolutional	3×3 (512)	$(N, 28, 28, 512)$	$(N, 56, 56, 512)$	$(N, 32, 32, 512)$	$(N, 16, 16, 512)$
	Convolutional	3×3 (512)	$(N, 28, 28, 512)$	$(N, 56, 56, 512)$	$(N, 32, 32, 512)$	$(N, 16, 16, 512)$
	Convolutional	3×3 (512)	$(N, 28, 28, 512)$	$(N, 56, 56, 512)$	$(N, 32, 32, 512)$	$(N, 16, 16, 512)$
	Max Pooling	-	$(N, 14, 14, 512)$	$(N, 28, 28, 512)$	$(N, 16, 16, 512)$	$(N, 8, 8, 512)$
Conv Block 5	Convolutional	3×3 (512)	$(N, 14, 14, 512)$	$(N, 28, 28, 512)$	$(N, 16, 16, 512)$	$(N, 8, 8, 512)$
	Convolutional	3×3 (512)	$(N, 14, 14, 512)$	$(N, 28, 28, 512)$	$(N, 16, 16, 512)$	$(N, 8, 8, 512)$
	Convolutional	3×3 (512)	$(N, 14, 14, 512)$	$(N, 28, 28, 512)$	$(N, 16, 16, 512)$	$(N, 8, 8, 512)$
	Max Pooling	-	$(N, 7, 7, 512)$	$(N, 14, 14, 512)$	$(N, 8, 8, 512)$	$(N, 4, 4, 512)$
Fully-connected Layer	Flatten	-	$(N,$ 25,088)	$(N,$ 100,352)	$(N,$ 32,768)	$(N,$ 8,192)
	Dense	-	$(N, 256)$	$(N, 256)$	$(N, 256)$	$(N, 256)$
	Dense	-	$(N, 2/3/4)$	$(N, 2/3/4)$	$(N, 2/3/4)$	$(N, 2/3/4)$

Table 5.3: Computational time (seconds) for different network input sizes

Procedure	**Large**	**Medium**	**VGG**	**Tiny**
Feature extraction	219.5	83.5	65.9	35.2
Training (per epoch)	3.98	1.52	1.26	0.63

shown in Table 5.3, the large size indeed costs significantly more computational time than that of the tiny size in the Conv operations.

Training configuration

As mentioned above, the network input size experiment was conducted only using the feature extractor by performing convolution computation once, refer to Fig. 5.4. As a supervised learning classification problem, the labeled data is split into two sets, one for training the model and the other for testing the generality. Since the total number of labeled data is 2,000, the ratio of the training and test sets was empirically set to 4:1, that is, 1,600 images for training and 400 ones for test. To avoid loss of generality for the prediction model, the label ratios (Fig. 5.3) in both training and test sets were kept the same, which, in this way, all tasks in training or test processes have the same data distributions.

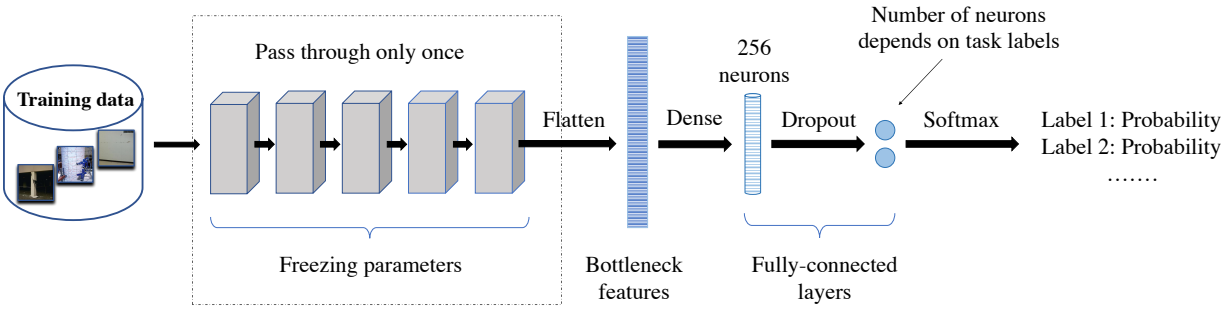


Figure 5.4: Configuration of the feature extractor for network input size experiment

For computational efficiency, no DA tricks were used in the feature extractor; instead, dropout was applied, which was shown to be very efficient for relieving over-fitting in deep neural networks [120]. Empirically, dropout rate is typically taken as about 0.5, but considering that the VGG-type model was pre-trained on a large dataset with over 15 million images and, on the other hand, the training dataset in this study only has 1,600 images, a heavy drop ratio of 0.7 was used to avoid over-fitting.

The training was carried out by optimizing the categorical cross-entropy loss, Eq. 2.5, using *Adam optimization* [53] based on the backpropagation algorithm, Eqs 3.7 & 3.8, which only requires first-order gradients with little memory requirements. β_1, β_2, and ϵ are three key parameters in Adam optimization, which were respectively assigned the empirical values 0.9, 0.999, and 10^{-8}. To avoid making a rapid convergence towards the pre-trained parameters, the learning rate η (Eq. 3.8) was set to 10^{-4}. Limited by the computational resources, the mini-batch approach was applied, which means instead of training with the whole dataset at one time, only a small batch with few sub-sets of the data was used for iterative training of the CNN. The amount of data in the batch, i.e., the batch size, was taken as 32 in this experiment. The number of training epochs was set to 25, and would be increased if the results do not converge (such as for the damage level and damage type tasks). For the feature extractor, the parameters initialization was conducted only in the fc-layers. The weight terms were sampled from a normal distribution with zero mean and 10^{-2} variance and biases were initialized with zeros. The evaluation criteria were the overall accuracy, Eq. 2.7, and the confusion matrix, Fig. 2.1.

Retrain ratio experiment

Network configuration

For simplicity, in the retrain ratio experiment, only default network input size 224×224 was used. Thus, the used network herein was exactly the same as that one used in the network input size experiment as listed in the fourth column of Table 5.2.

Instead of fine-tuning by layers, the model, which consisted of five Conv blocks (Table 5.2) was retrained via these Conv blocks in this experiment. Moreover, the study was conducted

by gradual comparisons, such as starting from only retraining the last block, then retraining the last two blocks and so on. Since no matter how many Conv blocks the model retrains in TL, the convolution computations in the forward propagation are inevitable, which require large computational resources and time. Moreover, the difference in the training effect is reflected in the number of parameters the model backpropagates. From Table 5.4, it is obvious that, except for the adaptation layers, most parameters were in the last several Conv blocks. The last two blocks plus the adaptation layers contained nearly 90% of the parameters of the whole network, and the last block plus the adaptation layers contained 64% of these parameters. However, the training time did not vary linearly with the number of parameters where even though only 10% of the parameters in the first frozen Conv block, much more computation time was needed to backpropagate the signal, Table 5.5. Moreover, considering the small dataset, retraining more Conv blocks leads to closer training of the whole network, which is more susceptible to over-fitting. Therefore, for convenience, only the effect of retraining the last one Conv block or retaining the last two Conv blocks were compared. Moreover, retraining more than three Conv blocks led to more severe over-fitting, which is discussed in a later section.

As mentioned above, one way to improve the prediction performance is by fine-tuning some parts of the network. Meanwhile, since the parameters of part of the CNN are adjusted by backpropagation, online DA can be applied to reduce over-fitting. This is beneficial to reduce the storage demand required for offline DA using the feature extractor.

Table 5.4: Number of trainable parameters for different retrain ratios in the backpropagation

Retrained blocks	**Component type**	**Spalling condition**	**Damage level**	**Damage type**
Fc-layers	6,423,298	6,423,298	6,423,555	6,423,812
One block + fc-layers	13,502,722	13,502,722	13,502,979	13,503,236
Two blocks + fc-layers	19,402,498	19,402,498	19,402,755	19,403,012
All	21,137,986	21,137,986	21,138,243	21,138,500

Table 5.5: Computational time (seconds) for different retrain ratios

Procedure	**Retrain one block**	**Retrain two blocks**	**Retrain all**
Training (per epoch)	128	133	160

Training configuration

For this fine-tuning experiment, the same training and test datasets as for the previous network input size experiment using feature extractor were used. Different from feature extractor, for each epoch, the training procedure went through the parameters in the partial unlocked Conv blocks and updated these blocks once. An online DA approach along with mini-batch were applied. During training, whenever a new batch arrived, all images in that

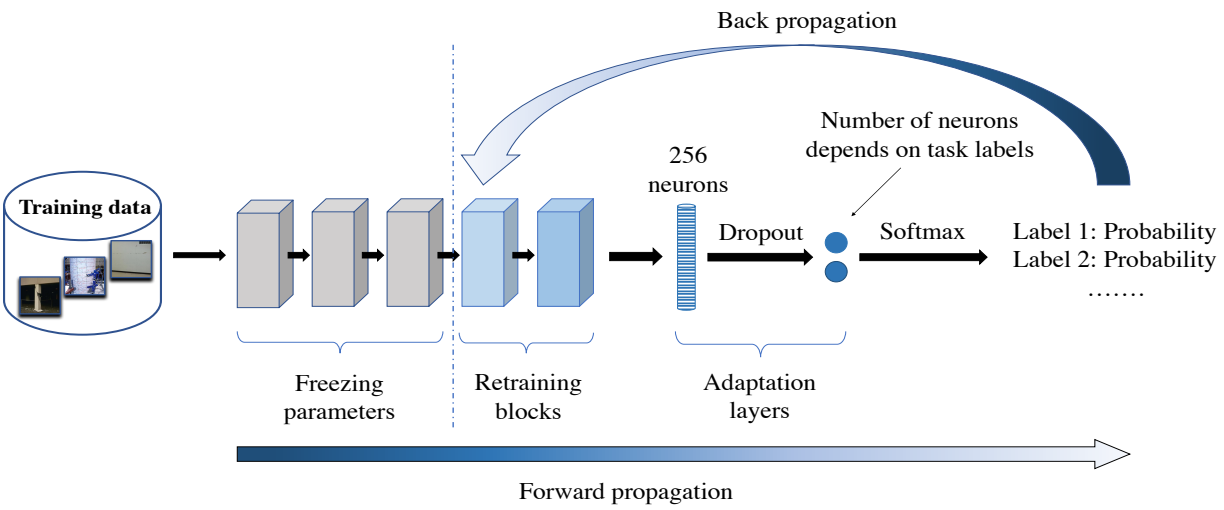


Figure 5.5: Configuration of the fine-tuning for the retrain ratio experiment

batch were passed into a transformation unit, which had random choices for zoom in/out, flip horizontally or vertically, rotate with random angles within five degrees, and translate in random directions (up/down/left/right). After such DA process, the input image was transformed to form another new image. These online DA tricks can generate more new data with more training epochs, which proved to be a quite efficient approach to deal with the over-fitting issues. Moreover, the online DA approach can save significant data storage memory requirements compared with the offline one, because the generated images only exist during the training process. After each training epoch, the generated images with DA are replaced by newly generated ones for the next epoch. It should be noted that even though new data can be created with an increasing number of training epochs, these generated data are highly correlated to the original ones. Thus, the online DA trick is viewed as an alternative approach to reduce the over-fitting of the model by training it with a larger dataset. However, for a better and more reliable performance, more real data is required to be collected.

In this experiment, some parameters are re-tuned in the Conv blocks. Thus, to avoid ruining the parameters from the pre-trained model, the learning schema differed from that of the feature extractor experiment. Herein, the recommended optimization method was to use *stochastic gradient descent* (SGD) *with momentum* considering a small learning rate [101]. Aiming to minimize the loss function L (taken as the categorical cross-entropy Loss, Eq. 2.5), SGD updates the model parameters W through moving in the direction opposite to the gradient. In that regard, the moving step size is controlled by the learning rate η. Moreover, another momentum term μ is added in the update step, Eq. 5.1. Analogous to the momentum in physics, this second momentum term works as a friction mechanism to avoid a rapid parameter change and helps the model achieve a better convergence with the momentum variable μ acting as the coefficient of friction.

$$W \leftarrow W - \eta \cdot \nabla_W L + \mu \cdot \Delta W \tag{5.1}$$

where $\nabla_W L$ is the gradient of the loss function with respect to the model parameters W and ΔW is the change of W. For this experiment, the learning rate η was set to 10^{-4} and the corresponding momentum variable μ was set to 0.9. Same batch size, dropout rate, etc. were used with similar settings as conducted in the previous network input size experiment.

Network comparison experiment

In this experiment, two ResNet architectures, ResNet-50 and ResNet-101, were implemented, and both were examined with feature extraction and fine-tuning. The comparisons were made first between these two different depths (50 and 101) of the ResNet. Subsequently, the best ResNet amongst these two depths and the VGG-type network were compared and discussed.

Network configuration

Similar to the above VGG-type network, the fc-layers after the original ResNet global average pooling (GAP) layer were replaced by two adaption layers with 256 neurons fc-layer and a softmax layer, whose output equals the number of classes. The remaining configurations of the network architectures in both ResNet-50 and ResNet-101 were taken same as the ones in [42]. As mentioned in Section 3.3, the first residual unit in each Conv block was set as the Conv unit to match the dimension between the Conv layers and the remaining residual units were all identity. The major difference between ResNet-50 and ResNet-101 is that the latter has more identity residual units in the Conv block 4 (which is applied 23 times instead of 6 times for the former, leading to additional $(23 - 6) \times 3 = 51$ Conv layers for a total of $50 + 51$ layers in ResNet-101), which might help with extracting mid- to high-level features. The detailed configurations are listed in Table 5.6, where N represents the number of data, i.e., 2,000 in ϕ-Net-2000 used in this chapter.

Training configuration

While implementing the feature extractor, all parameters were fixed in the Conv blocks, and the output tensor before the fc-layer (Fig. 5.4) was extracted as features. The weights were initialized from a normal distribution with zero mean and 10^{-2} variance (biases are initialized with zero) and optimized the categorical cross-entropy loss using Adam method based on backpropagation, where three key parameters β_1, β_2 and ϵ were, as before, assigned with empirical values 0.9, 0.999 and 10^{-8}. The learning rate η was set to 10^{-3} and manually changed to 10^{-4} after 200 epochs of training.

Since the major difference of ResNet-50 and ResNet-101 is the number of identity residual units in block 4, while adopting fine-tuning in this experiment, only either Con block 5 or Conv blocks 4 & 5 were tuned for both ResNet-50 and ResNet-101 considering the concerns of over-fitting due to a small training dataset. The categorical cross-entropy loss was optimized

Table 5.6: Configurations of ResNet-50 and ResNet-101

Block	Output shape	ResNet-50	ResNet-101
Input	$(N, 224, 224, 3)$	-	
Conv Block 1	$(N, 112, 112, 64)$	7×7 (64), stride 2	
Conv Block 2	$(N, 56, 56, 256)$	Max Pooling, filter 3×3, stride 2	
		$\begin{bmatrix} 1\times1\ (64) \\ 3\times3\ (64) \\ 1\times1\ (256) \end{bmatrix} \times 3$	
Conv Block 3	$(N, 28, 28, 512)$	$\begin{bmatrix} 1\times1\ (128) \\ 3\times3\ (128) \\ 1\times1\ (512) \end{bmatrix} \times 4$	
Conv Block 4	$(N,$ 14, 14, 1,024)	$\begin{bmatrix} 1\times1\ (256) \\ 3\times3\ (256) \\ 1\times1\ (1024) \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1\ (256) \\ 3\times3\ (256) \\ 1\times1\ (1{,}024) \end{bmatrix} \times 23$
Conv Block 5	$(N,$ 7, 7, 2,048)	$\begin{bmatrix} 1\times1\ (512) \\ 3\times3\ (512) \\ 1\times1\ (2{,}048) \end{bmatrix} \times 3$	
Global Average Pooling	$(N,$ 2,048)	-	
Fully-connected Layer	$(N, 256)$	Activated by ReLU	
Fully-connected Layer	$(N, 2/3/4)$	Activated by Softmax	

by SGD with momentum in an exponential decay learning schedule ($\eta_t = \eta_0 \cdot e^{-d \cdot t}$), where the momentum variable μ was 0.9 and the initial decay value η_0 and decay rate d were 10^{-4} and 10^{-6}, respectively. Moreover, to better alleviate the negative influence of over-fitting, online DA was also applied during training, whose setting was the same as that in the retrain ratio experiment.

5.4 Experimental Results & Discussions

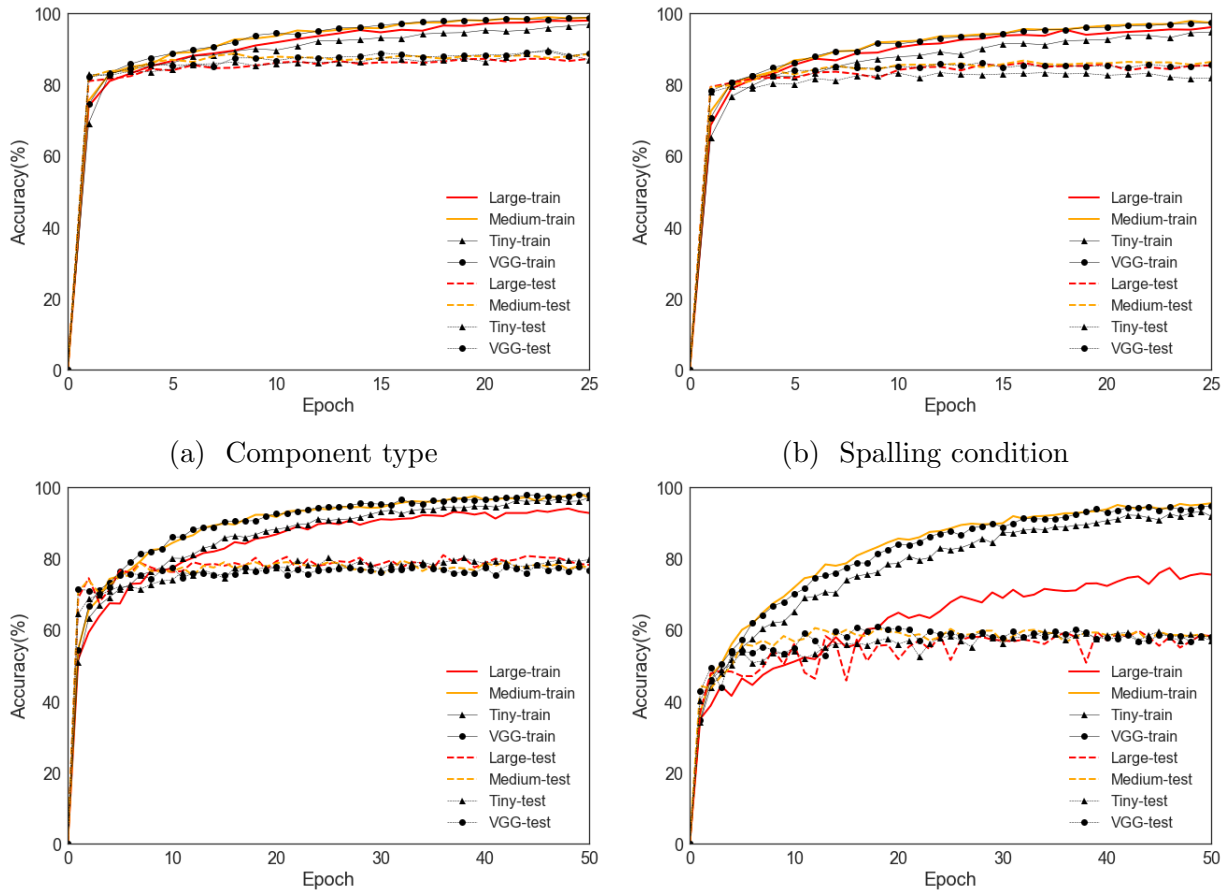
Network input size experiment

Figs 5.6a to 5.6d plot the history of the accuracy of both training and test processes for all four detection tasks. It is noted that in damage level and damage type tasks, more training epochs are needed for convergence, requiring an upper limit of 50 epochs instead of the specified 25 used for the other two binary tasks. The first observation is that computations converge quickly at the very beginning, which is due to the fact that extracted features are fixed and the training process is only working on different nonlinear combinations of the

features to reduce the loss with a small size fc-layers. However, with poor or non-optimal features, it is also easy to converge to local minima making it difficult to escape these local minima, which should be carefully handled.

The model using tiny input size performed worse than the other three due to lower resolution, which leads to lack of information. This phenomenon can also be attributed to insufficient feature, which makes it harder to achieve a good training accuracy with fewer training epochs, let alone lower test accuracy. On the contrary, the models using input size with medium, large, or VGG size performed well and also similarly for the binary tasks. However, inputting an image with higher resolution with a large input size did not perform better than a medium resolution one or a VGG size for multi-class tasks. This observation is attributed to the fact that more parameters need to be trained with a large model caused by a large input. The VGG size seems to be the relative optimal one amongst the four considered size cases for all tasks. Therefore, it is concluded that the VGG-type network receiving an image with size around 224×224 can provide enough information to handle the considered four tasks, with which the CNN model can achieve good and stable results. The model with too large (i.e., costly) or too small (i.e., less informative) input size would probably perform worse. Table 5.7 lists the best accuracy for the four detection tasks from the VGG size input. Clearly the results for the binary classifications are very promising considering the computation time and the possibility of using a small dataset. Meanwhile, even though there are severe over-fitting problems in the multi-class tasks, compared with the lower bound accuracy of 33.3% and 25.0% using random guesses of the respective tasks with three and four classes, the obtained accuracy of 77.0% for the damage level task (3 classes) and 57.7% for the damage type task (4 classes) are reasonably acceptable considering the used small-scale dataset.

In general, the results reflect some degree of over-fitting. For the two binary tasks, it is observed that there is about 10% over-fitting (obtained as the difference in accuracy between the training and test). On the other hand, over-fitting gap values were around 20% and 37% for the three-class and four-class tasks, respectively. In all tasks, the more difficult the task, the more severe the over-fitting. Since training accuracy in all tasks can reach over 90%, one concludes that the main reason for over-fitting is due to less training data because the CNN architecture already holds enough complexity to classify the problem accurately. Thus, the CNN loses the generality to extend the learned parameters from a small training dataset to the unseen test dataset. In other words, this is equivalent to the generality issue of the extracted features, attributed to the fixed Conv blocks. Therefore, even though low-level features of the VGGNet might be common and sharable, due to difference between the source and the target domains, the high-level features from the source domain could be incompatible with those of the target domain. Thus, there are three ways to improve the performance of the model to be pursued in future studies: (1) increase the training dataset, (2) use the same architecture but fine-tune some parts of the CNN, and (3) change to a better architecture. As mentioned above, labeled training data are usually expensive to obtain. Therefore, fine-tuning and implementing other network architectures are more feasible choices amongst the three listed above.



(a) Component type (b) Spalling condition

(c) Damage level (d) Damage type

Figure 5.6: Accuracy history of four recognition tasks using feature extractor for the network input size experiment

In predictive analytics, the classification accuracy for true prediction alone may not be very reliable if the dataset is unbalanced or there are more than two classes in the dataset, but confusion matrix (CM) [56] can give a better way to evaluate the results. Therefore, normalized CM was computed for the classifier of the VGG size input experiment, which presents the probability of correct and incorrect predictions with values broken down by each class. In Fig. 5.7, both binary tasks performed quite well with high correct prediction accuracy and low misclassification error. However, in the spalling condition task, 20% of "spalling" images were misclassified as "non-spalling", a finding that should be carefully assessed and possibly attributed to some deficiency in the detection of the change of texture (indicative of spalling condition) in these misclassified images. In the damage level task, "undamaged state" and "moderate to heavy damage" achieved better results than "minor damage", which can be explained by the existence of sharp change of pixels within the neighboring areas, namely an edge [104] was more obvious in the two extreme classes compared with the intermediate

Table 5.7: Best accuracy results (%) in the network input size experiment

Task	Number of classes	Training accuracy	Test accuracy
Component type	2	99.1	88.8
Spalling condition	2	97.1	85.4
Damage level	3	97.5	77.0
Damage type	4	94.2	57.7

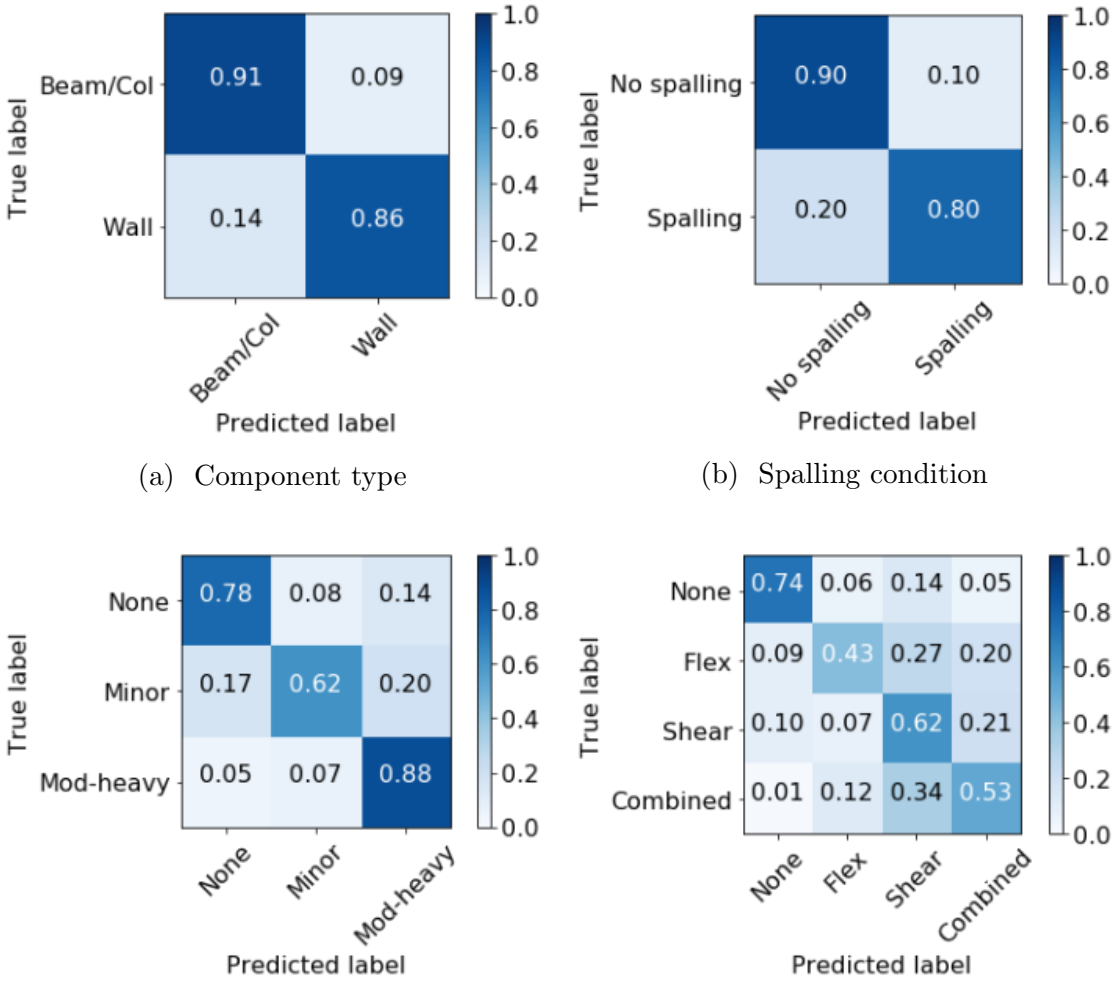
class. Finally, in the damage type task, the prediction of the undamaged state still behaved well but its accuracy slightly decreased due to the increase of the number of classes from 3 to 4. Moreover, because the "flexural damage", "shear damage", and "combined damage" classes have some similarities, which may confuse the classifiers, they are more prone to misclassification.

The detection results through the feature extractor are quite promising even with some misclassifications. The results indicate the great generalization of the VGG-type architecture, and the recognition accuracy can clearly be improved with increasing the training data or by using more parameters in the CNN architecture through fine-tuning. The most computation time was consumed in the feature extraction procedure which was only performed once, leading to a very fast training for the classifier (Table 5.3). Thus, in large projects, feature extractor can be used in preliminary analysis before running the costly computations on the complete CNN.

Retrain ratio experiment

With more parameters, the fine-tuning model converges after more rounds of training than that of the feature extractor. Thus, the history of loss becomes an important indicator to evaluate the convergence and to learn about the performance of the model. Figs 5.8a to 5.8d plot the comparisons of the loss and accuracy histories between retraining one block and two blocks.

Considering over-fitting issues, fine-tuning one Conv block performed well which implies that no over-fitting occurred and slightly over-fitting took place for fine-tuning two blocks especially for the four-class classification task. It is noted that in Figs 5.8a for component type and 5.8c for damage level, the test accuracy is slightly higher than the training accuracy for the case of retaining two blocks, which is explained by the effect of the dropout. When the dropout was applied, during training some neurons were randomly dropped, and during testing no neurons were dropped, which means that the trained network is less complex than the complete one used or the test. This is analogous to *learn harder and then test easier*. The over-fitting issues in retraining two blocks are still due to the lack of data since more parameters need to be determined. Thus, there is no need to retrain more than two blocks, e.g., three or more blocks in this case, which is expected to definitely have more severe over-fitting problems.



(a) Component type

(b) Spalling condition

(c) Damage level

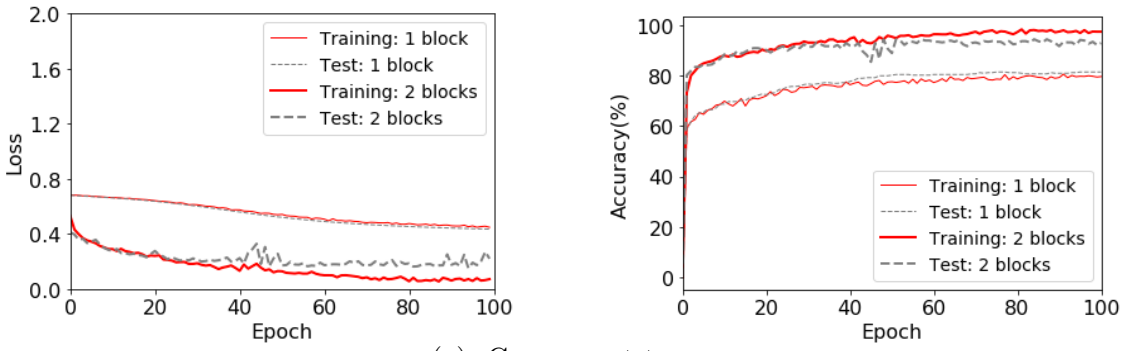
(d) Damage type

Figure 5.7: Normalized confusion matrices of the test prediction using feature extractor for the network input size experiment

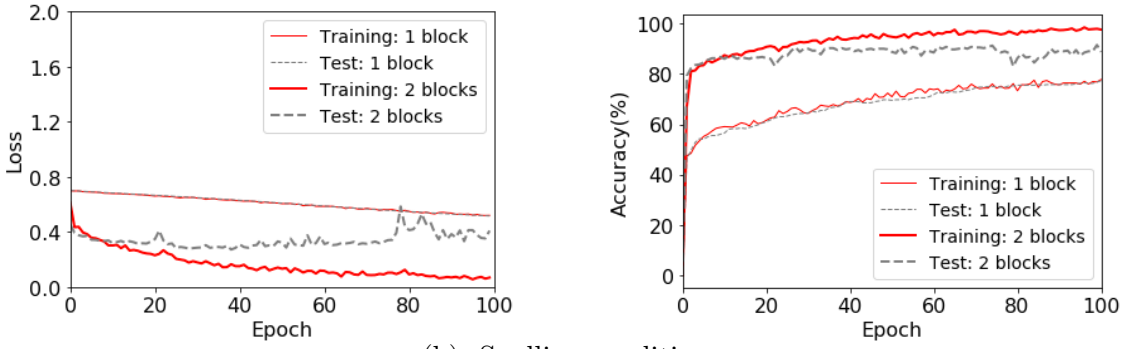
The loss is usually not used as a dominating metric (e.g., overall accuracy) to evaluate the goodness of the model, and few discussions are available about the value of the loss in most previous studies. However, the history and trend of the loss can still be used for understanding and monitoring the supervised learning process. If the loss is decreasing, it shows that the network is learning as intended. For the two binary tasks (component type and spalling condition) with retraining one block, both training and test losses are somewhat continuing to decrease with roughly 36% and 26% linear reduction rates (as the estimated slop of the loss curves), respectively, where the losses are still not sufficiently close to zero even after 100 epochs, which indicates that there still exists room for model performance improvement. However, for the two multi-class tasks in one block, following a big drop, both training and test curves become flat where the linear reduction rate of the losses decreases to 20% and 7%,respectively, while the losses are still high (close to 1.0 and 1.3 after 100 epochs for the damage level and damage type tasks, respectively). This observation indicates that it is hard to obtain more improvement with further training based on the current parameter complexity. While in retraining two blocks, training loss curves behave very well for all tasks where the losses are much lower between 0.1 and 0.25 for all tasks, but the test losses do not decrease and instead oscillate after about 20 epochs with the curves becoming jagged. This is evidence of over-fitting indicating that the network has enough complexity to learn some features but lacks the generality to extend these features to new datasets due to the small size dataset used for training. In general, even with some degree of over-fitting, the models with retraining two blocks have lower losses than those with only retraining one block (ignoring some exceptions during small intervals of oscillations near the 80th epoch for the spalling condition task), implying a better performance. In addition, except for the damage type task and to a lesser extent the spalling condition task, the accuracy results indicate that some degree of over-fitting is still acceptable for retraining two blocks. From the perspective of accuracy, which is the most important indicator to evaluate the models, the accuracy for both training and test shows significant improvement from retraining one to two Conv blocks for all four detection tasks, especially for the damage level detection, which shows that it is efficient to learn features of damage level by retraining more parts of the network. Regardless of the training accuracy, the test accuracy for the component type, the spalling condition, and the damage level reaches about 90% or more which is very promising considering the use of such a small dataset. Even for the damage type task, 68.8% test accuracy by two blocks is more than 25% higher than that for one block (with accuracy of about 40% only), and much higher than the 25% accuracy, which is for the random guess for a four-class classification problem.

From the above analysis, retraining one block leads to under-fitting but stable performance. On the contrary, retraining two blocks leads to over-fitting but great improvement of accuracy. Therefore, it is believed that for the current dataset and tasks, more important parameters representing structural features exist in the second to last Conv blocks. By increasing the dataset size, it is expected that more studies can be performed by fine-tuning more Conv blocks to achieve even better performance.

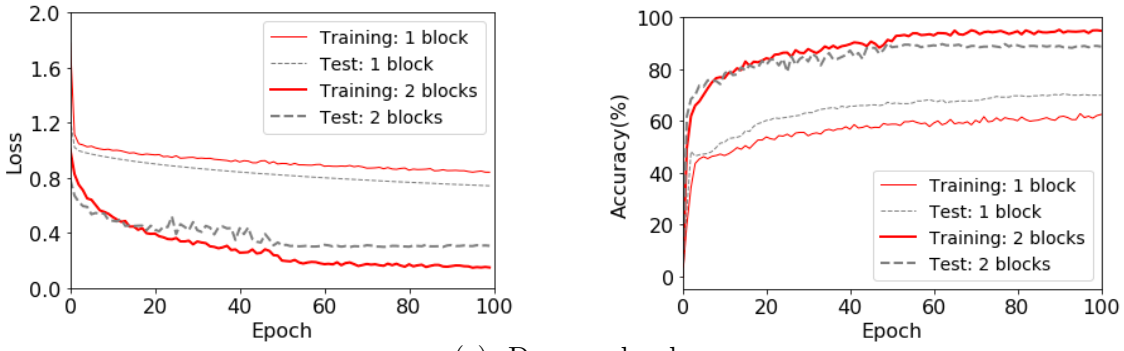
The model with retraining two Conv blocks has better performance than only retraining



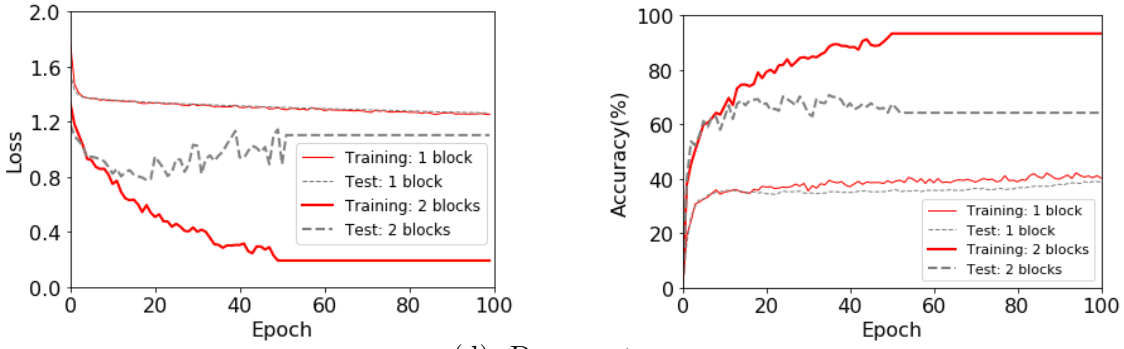
(a) Component type



(b) Spalling condition



(c) Damage level



(d) Damage type

Figure 5.8: Loss and accuracy history of the four recognition tasks using fine-tuning for the retrain ratio experiment

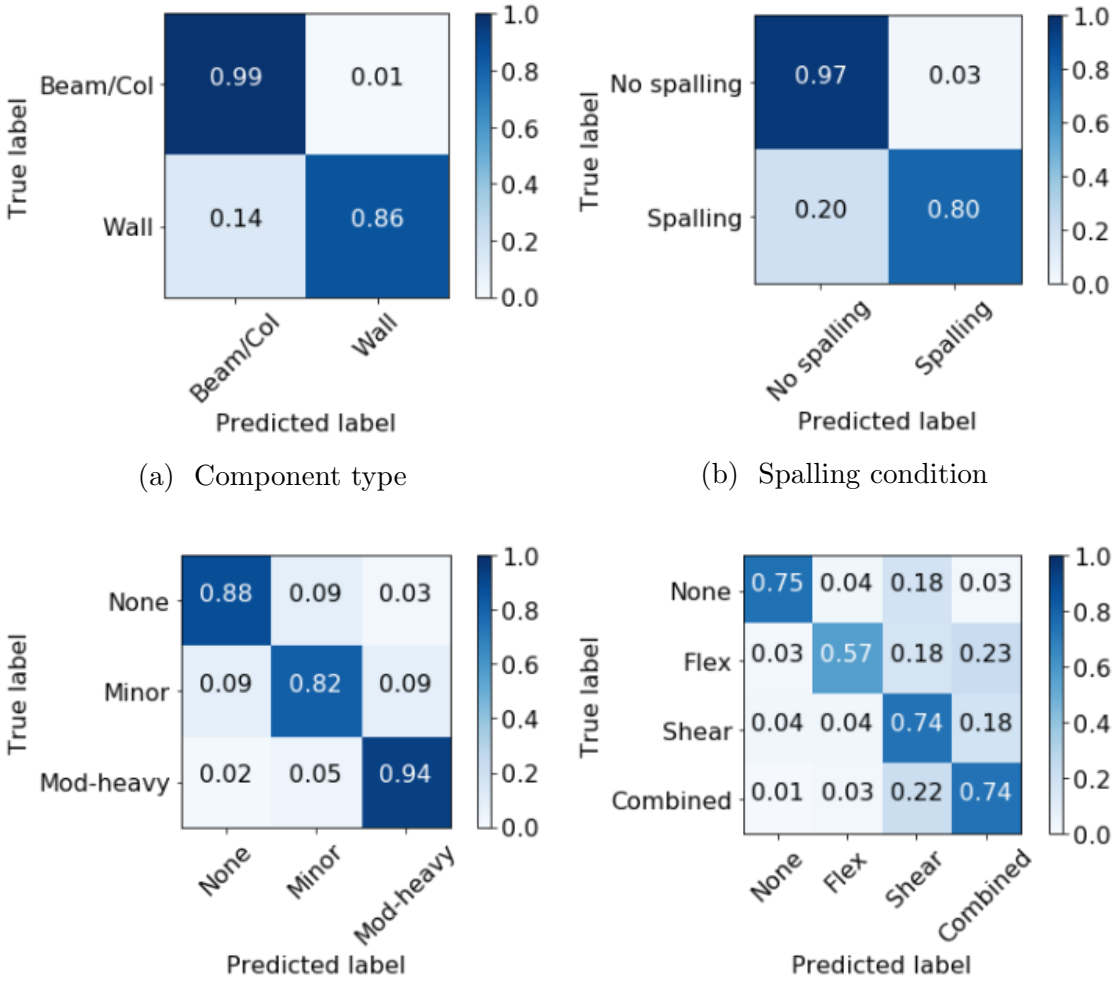
one block, and the best prediction accuracy is listed in Table 5.8. Compared with the results obtained by applying the feature extractor, the fine-tuning approach reduced the over-fitting making use of the DA and adjustment of the mid- to high-level features, including greatly improving the recognition accuracy. For the binary tasks, the test accuracy by fine-tuning two blocks improved by 5.7% to 6.1%. More interestingly, the accuracy for the damage level improved by 12.7% with only 5.6% over-fitting, which shows the good properties when dealing with multi-class classification problems. In addition, even though for the hardest damage type task (four classes), the model with retraining two blocks improved the accuracy by 11.1% reaching 68.8% and reduced over-fitting from 36.2% down to 23.0%. Moreover, the normalized CM was computed for the fine-tuning experiment, Fig. 5.9. Compared with the feature extractor, the recognition accuracy for the "beam/column" and "non-spalling" classes increased slightly, but the misclassification for "spalling" still has the same value, which indicates the need for more training data. For the multi-class tasks, the accuracy has significantly improved, especially to identify the "minor damage" class from the "moderate to heavy damage" one. However, for the damage type, there still exists around 20% misclassification error among the three types of damage, which should be evaluated carefully in future studies. Thus, by retraining more parameters and invoking the DA, the classifier behaved better than the one based on the feature extractor in all ways except for the training time (compare Table 5.3 to Table 5.5 where the training time per epoch for the case of retaining two blocks (133 sec.) significantly increased by two orders of magnitude (a factor of about 105) compared to the case of the feature extraction using the VGG input size (1.26 sec.)), which is a trade-off between performance accuracy and computational cost.

Table 5.8: Best accuracy results (%) in the retrain ratio experiment

Task	**Classes**	**Training accuracy**	**Test accuracy**
Component type	2	98.4	94.5
Spalling condition	2	98.5	91.5
Damage level	3	95.3	89.7
Damage type	4	91.8	68.8

Network comparison experiment

The computation results of ResNet-50 and ResNet-101 using feature extractor with 224×224 input size are shown in Fig. 5.10. It is observed that all tasks converge fast and their performance did not significantly improve even with adjusting to lower learning rate after 200 training epochs. This observation is attributed to the fixed features extracted by the Conv blocks. Another observation is that the ResNet-50 and ResNet-101 shared similar performance, both prediction accuracy of the two binary tasks reached nearly 80% and reached about 66% for the damage level task without over-fitting. However, in the damage type task, the accuracy for both of ResNet-50 and ResNet-101 were unsatisfactory being about



(a) Component type

(b) Spalling condition

(c) Damage level

(d) Damage type

Figure 5.9: Normalized confusion matrices of the test predictions by fine-tuning two Conv blocks from the retrain ratio experiment

40%, which is not much higher than the 25% of the random guess of four-class classification. This issue is attributed to less information and inadequate features extracted by the fixed Conv blocks. Compared with the VGG-type networks, the accuracy of both ResNets are lower, which is attributed to the fact that the pre-trained Conv blocks in the ResNets may have a weaker generalization than that of the pre-trained blocks in the VGG-type network, and the extracted features by these blocks of the source domain may not be well-aligned with the new recognition tasks in target domain. However, the performance of the ResNets is expected to be improved via fine-tuning.

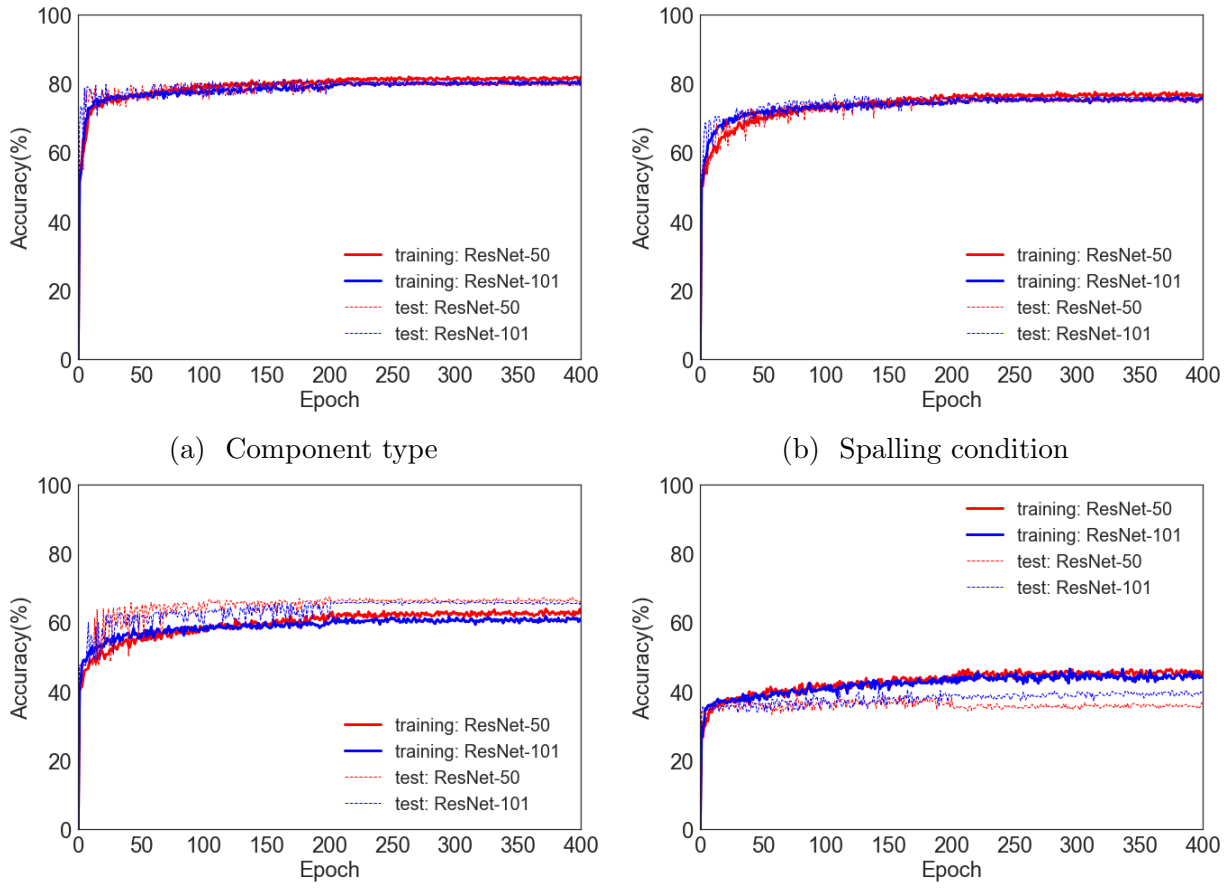
Regarding the computational efficiency, the feature extractor only consumes small amount of the computational time, roughly 28 and 48 sec. for the feature extraction process of ResNet-50 and ResNet-101, respectively, and only 0.2 sec. for both networks in each training epoch. Thus, training the ResNets is much faster (i.e., $1.26/0.2 \approx 6$ times faster) than the VGG-type network (Table 5.3). Since the performance of the feature extractor for ResNet-50 is comparable to that of ResNet-101, ResNet-50 with feature extractor can be considered for preliminary analysis for the component type identification, the spalling check, and the damage level evaluation. On the other hand, for the damage type task, more studies should be conducted to mitigate the misclassification risks.

While adopting the fine-tuning strategy, it is shown that it greatly improved the prediction performance for both ResNets, and the results are shown in Fig. 5.11. The slight differences between the two networks can be observed. The accuracy of first three tasks reached over 90%, and for the damage type task, the test accuracy increased over 70% with about 20% over-fitting. This observation also indicates that for the current small dataset, ResNet-50 already holds sufficient complexity for the recognition tasks in addition to its computational efficiency (roughly 82 sec. per epoch for ResNet-50 and 160 sec. for ResNet-101). Another interesting observation is the plateau in the test accuracy history at the beginning, i.e. within the first 10 epochs, which is attributed to incompatible initial parameter values and large learning rate. With online DA and decaying learning rate, the model escaped this plateau within about 5 to 10 additional training epochs, and finally achieved very promising results for all four tasks.

Table 5.9: Best test accuracy results (%) of variant VGGNet, ResNet-50 and ResNet-101 in the network comparison experiment

Task	**Classes**	**VGGNet**	**ResNet-50**	**ResNet-101**
Component type	2	94.5	93.5	94.0
Spalling condition	2	91.5	88.5	92.0
Damage level	3	89.7	90.5	91.5
Damage type	4	68.8	72.0	74.5

In general, ResNet-101 using fine-tuning achieved the best performance (test accuracy) in all four tasks among all considered models, and the best results of all models are list in Table 5.9. Comparing its results with the previous VGGNet cases, ResNet-101 outperforms



(a) Component type

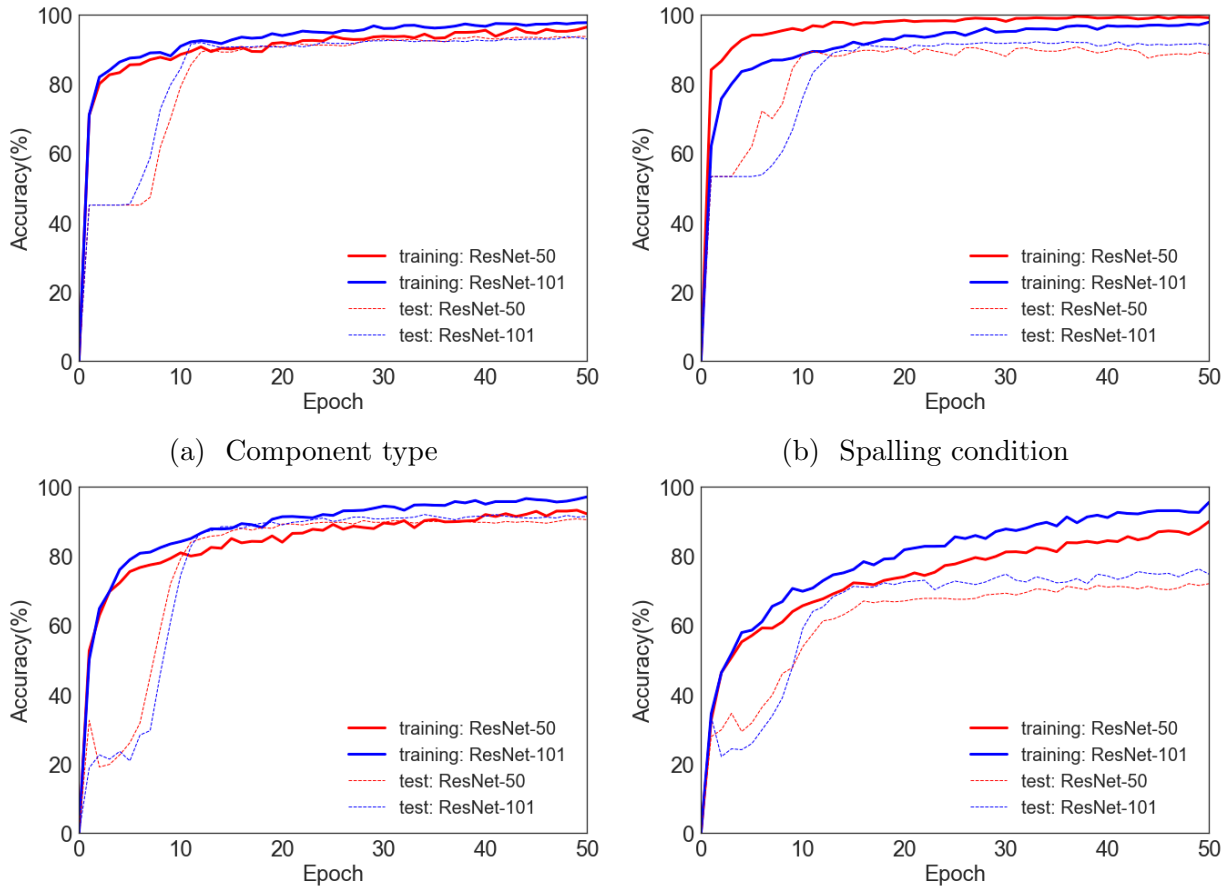
(b) Spalling condition

(c) Damage level

(d) Damage type

Figure 5.10: Accuracy history of ResNet-50 and ResNet-101 using feature extractor for the network comparison experiment

the VGG-type network in the last three tasks and has a good improvement in the hardest task (damage type), where it improves the test accuracy by 5.7%. For the component type task, ResNet-101 is only 0.5% lower than VGG, which is acceptable given the already very high accuracy of 94%. In addition, ResNet-50 achieved equivalently good results compared with VGGNet, where it has higher test accuracy in the harder multi-class tasks (damage level and damage type), and slightly lower accuracy in the easier binary tasks. Regarding the computational efficiency, the time used in fine-tuning two Conv blocks in ResNet-101 (160 sec. per epoch) is the same as retraining the whole VGGNet (refer to Table 5.5). However, due to more trainable parameters in ResNet-101, such time increment is not large and still acceptable. Moreover, ResNet-50 only spent 82 sec. per epoch, which is much less than that of the VGGNet cost when retaining two blocks (i.e., 133 sec. per epoch from Table 5.5). Thus, the family of ResNets indicates a competitive performance in the computational efficiency and classification accuracy, where the fine-tuned ResNet-101 is the relatively optimal model



(a) Component type

(b) Spalling condition

(c) Damage level

(d) Damage type

Figure 5.11: Accuracy history of ResNet-50 and ResNet-101 using fine-tuning for the network comparison experiment

for the considered four recognition tasks using the ϕ-Net-2000 dataset.

5.5 Summary

In this chapter, the working principle of TL is discussed first. This is followed by conducting the numerical experiments of using TL on the small-scale ϕ-Net-2000 dataset. Three major experiments with respect to (1) network input size, (2) retrain ratio, and (3) comparison between different networks were conducted. Conclusions from these experiments are listed as follows:

- The relative optimal network input size is 224×224, which was used in training prototype VGGNet and ResNet, and inadequate size would probably perform worse due to either lack of information or computational complexity and inefficiency.

- Some parameters in the last two Conv blocks represent structural damage features. In the retrain ratio experiment, VGG-type network with retraining two Conv blocks achieved very promising accuracy of about 90% for the binary and the three-class tasks. Due to the complexity of the the four-class task and the use of a small training dataset, about 68% test accuracy with 23% over-fitting gap value was obtained, which is much higher than what is expected from the lower bound random guess for a four-class classification problem.

- Both ResNet-50 and ResNet-101 computations are fast under feature extractor but with similar yet limited accuracy especially for tasks with numbers of classes larger than 2. However, the performance of both ResNet-50 and 101 can be significantly improved through fine-tuning, where ResNet-101 has higher test accuracy than that of ResNet-50. When compared to the best variant of the VGGNet, the fine-tuned ResNet-101 outperforms it in most detection tasks, and it is considered as a better model. Moreover, ResNet-50 is thought to be of a similar accuracy level with the variant VGGNet, but it is more computationally efficient.

In conclusion, TL is an effective training approach to mitigate the data deficiency in vision-based SHM tasks. Its feature extractor strategy acts as a preliminary and computationally efficient analysis with an reasonable recognition accuracy on the target dataset. On the other hand, fine-tuning can be considered as an improvement tool beyond the feature extractor if good computation resources are available. All considered experiments in this chapter indicate the good generalization properties and transferability of VGGNet and ResNet to structural image datasets, which validate the effectiveness and high potentials of deep CNN in vision-based SHM. These observations provide the foundation for conducting more complex benchmark tasks using the complete ϕ-Net dataset in Chapter 6.

Chapter 6

Benchmark Experiments on the ϕ-Net

Based on the ϕ-Net dataset discussed in Chapter 4 and the preliminary experiments in Chapter 5, a comprehensive investigation applying DL in general tasks of vision-based SHM is explored in this chapter. Similar to Chapter 4, this chapter starts with the experimental objectives, followed by the experimental setup, and finally the experimental findings of the considered eight tasks.

6.1 Experimental Objectives

In Chapter 5, through preliminary experiments using deep CNN with the TL strategy on a reduced version of ϕ-Net dataset, namely the ϕ-Net-2000 dataset, the effectiveness of DL for specific vision-based SHM problems has been evaluated. This revealed the high potential of using DL in more complex scenarios as explored in this chapter.

The purpose of this chapter is to provide benchmarks and reference performance through conducting comprehensive experiments on all eight benchmark tasks using the full ϕ-Net dataset. Here, benchmark experiments were conducted with three well-known deep CNN models, namely VGG-16, VGG-19 [112], and ResNet-50 [42] with overall accuracy as the evaluation metric, Eq. 2.7. From Fig. 4.13, some tasks contain fewer images, e.g., the collapse mode task has only 1,226 training images, representing a data deficit case where a regular training approach may not work. Thus, two common strategies were explored to address this issue, namely data augmentation (DA) and transfer learning (TL). Besides using VGG-16, VGG-19, and ResNet-50 models, four comparison cases were conducted for each task and model as listed below:

1. Train the model from scratch, denoted as baseline model (BL model).
2. Train the model from scratch with DA (BL-DA model).
3. Fine-tune (making use of TL) the pre-trained model from ImageNet (TL model).

4. Fine-tune (making use of TL) the pre-trained model from ImageNet with DA (TL-DA model).

6.2 Experimental Setup

Network configuration

Based on the conclusions in the network input size experiment in Chapter 5, the input size for all models remained the same as 224×224. However, several other network architecture modifications were made to reduce the difference between the ImageNet dataset and the ϕ-Net dataset. These are summarized as follow:

- For VGG-16 and VGG-19 models, fc-layers after Conv blocks were replaced by three adaptation layers, which have 256 and 128 neurons in the first two layers, respectively. Moreover, nonlinear ReLU activation function was used in these two layers. The number of neurons in the last adaptation layer (softmax layer) was adjusted to satisfy the number of classes in each task.
- For ResNet-50 model, fc-layers after Conv blocks were replaced by two adaptation layers, where the first layer has 1,024 neurons and is activated by ReLU. Similarly, the last softmax layer was modified according to the number of classes in each task.

Training configuration

For a fair comparison, the training configurations for each task were fixed. From some preliminary convergence analysis, these settings were 50 training epochs, 64 batch size, and test accuracy computed at the end of each training epoch. The stochastic gradient descent (SGD) was applied along with a piece-wise decayed learning rate for all cases with the following learning rate schedule: 10^{-2} (first 20 epochs), 10^{-3} (21-30 epochs), 10^{-4} (31-40 epochs), 10^{-5} (41-45 epochs), and 10^{-6} (the remaining epochs). The intuition behind this learning rate schedule is that in the early stages of training, a larger learning rate is helpful to avoid having the model trapped in local minima, and the model is expected to somewhat freely explore. After training for the early epochs, the learning rate decreases where such smaller learning rate stabilizes the model and avoids large oscillations. To avoid over-fitting, dropout was applied in the adaptation layers with a dropout rate of 0.5.

In cases (3) and (4) using TL, the model parameters in the Conv layers were loaded from the ImageNet pre-trained models[1], and the parameters in the following adaptation layers were initialized randomly. On the other hand, in cases (2) and (4) with DA, transformation was performed from random combinations for six possibilities: (a) horizontal translation within 10% of the total width, (b) vertical translation within 10% of the total height, (c)

[1] `https://keras.io/applications/`.

rotation within 5 degrees, (d) zoom in less than 120% of the original size, (e) zoom out less than 80% of the original size, and (f) horizontal flip.

Due to the limitations of the computing resources, all cases were only performed once except some cases where the training loss did not change in the first 10 epochs. More experiments are planned to be conducted in the future. The implementation of the benchmark experiments was conducted on TensorFlow and Keras[2] platforms and performed on CyberpowerPC with single GPU (CPU: Intel Core i7-8700K@3.7GHz 6 Core, RAM:32GB & GPU: Nvidia Geforce RTX 2080Ti).

6.3 Experimental Results & Discussions

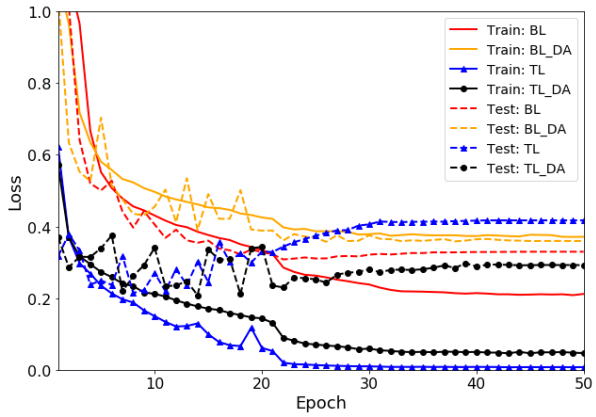
Task 1: Scene level

Results for test accuracy and training history of Task 1 (scene level) are shown in Table 6.1 and Fig. 6.1. Task 1 is a three-class classification, where the random guess is 33.3%. From Table 6.1, BL achieved promising results ($\sim$ 87%), which is significantly better than the random guess. There is not much difference between the different DL models, but ResNet-50 performed slightly better than the other two, achieving the best test accuracy of 93.4%. Because of sharing a similar architecture, the performance of VGG-16 and that of VGG-19 were similar. The BL-DA performance was worse than that of the BL model. Similarly, the TL-DA performance was worse than that of the TL model. In general, DA did not significantly contribute to improving the results, and the results were even worse when training from scratch. On the contrary, the TL worked well and clearly improved the performance by 6% for the best model. Figs 6.1b, 6.1d, and 6.1f present a clear view for the whole accuracy history whereby it is obvious that models using TL outperformed those that did not use TL. Moreover, the training history curves of the models using TL are smoother and more stable than those that did not use TL, especially for the ResNet-50 model. The results clearly indicate that the TL provides a better initialization and more stable training environment than training from scratch.

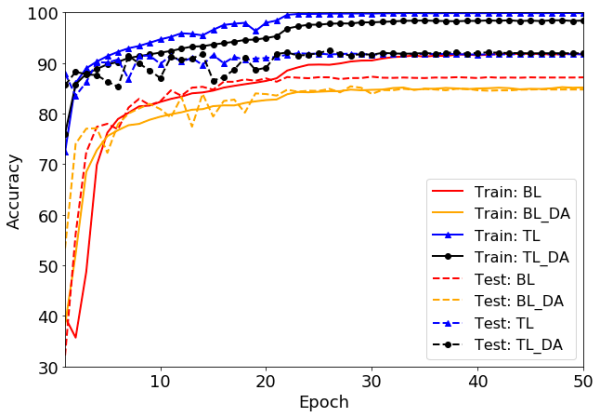
Table 6.1: Test accuracy results (%) for each case in the scene level task

Model	BL	BL-DA	TL	TL-DA
VGG-16	87.3	85.4	91.9	92.5
VGG-19	87.4	84.0	92.1	92.1
ResNet-50	87.7	88.0	93.4	93.0

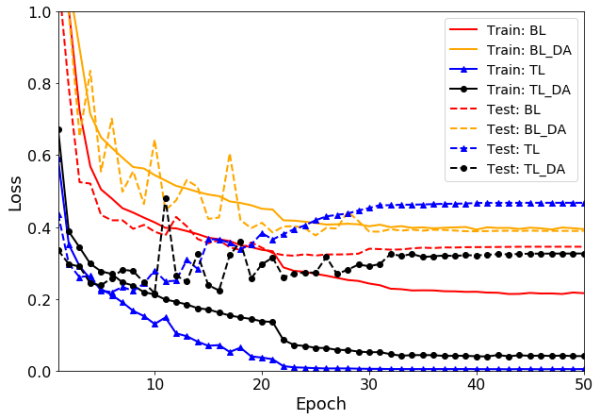
[2] `https://keras.io/`.



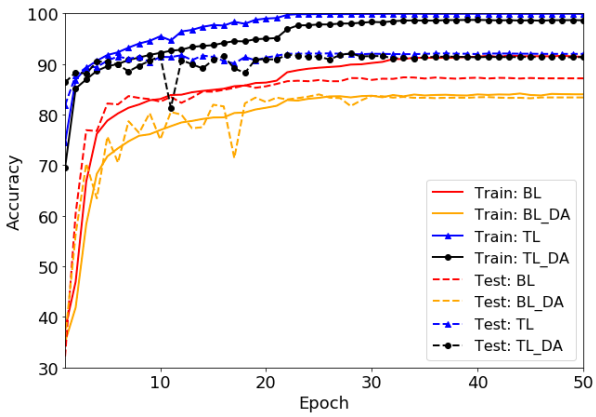
(a) Loss history of VGG-16



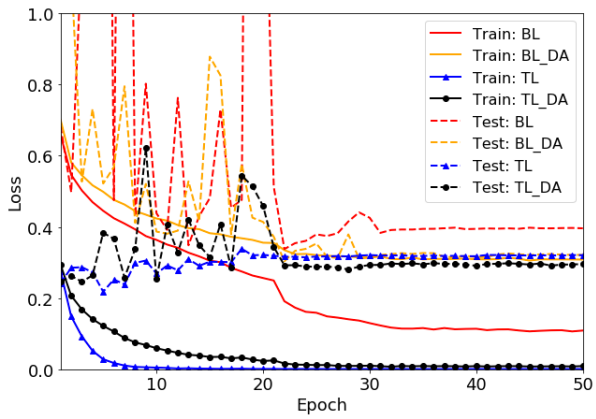
(b) Accuracy history of VGG-16



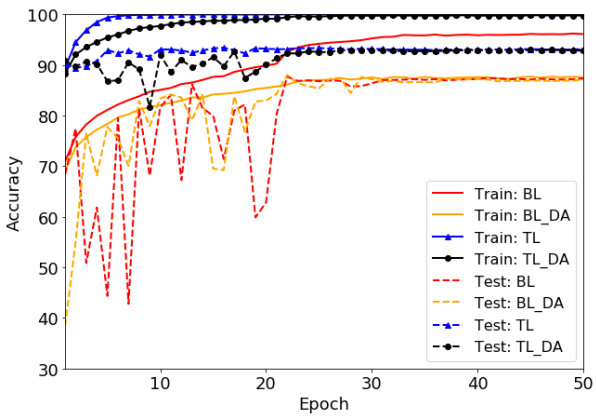
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.1: Training history of DL models for 50 epochs in the scene level task

Task 2: Damage state

Results for test accuracy and training history of Task 2 (damage state) are shown in Table 6.2 and Fig. 6.2. Task 2 is a binary classification where the random guess is 50%. As another basic and common detection task, the best model achieved 88.9%, which is approximately close to perfect performance. The best result was achieved by ResNet-50 using both TL and DA. Note that in this experiment under the TL setting, DA improved the performance slightly, in contrast to that observed in Task 1. A more interesting observation here is that compared to ResNet-50, VGGNets training from scratch did not produce stable and acceptable results ($\sim$63% and 61%). The test accuracy was just barely higher than a random guess where such results are not acceptable for sensitive damage detection. Thus, in Task 2, training from scratch only is far from acceptable. On the contrary, the best performances of each model that adopted DA and TL were close, with ResNet-50 using TL-DA slightly better. Such observations indicate that all models have high potential if trained using appropriate strategies.

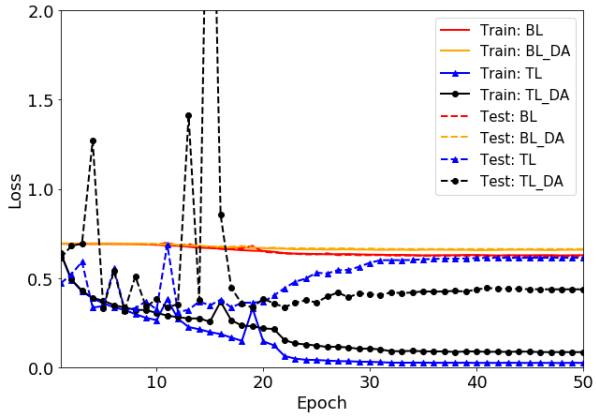
Fig. 6.2 provides more information for the effectiveness of TL, which cannot be reflected in Table 6.2. All three models indicate similar trends in training. The accuracy of training the models from scratch remains trapped at an early stage even with a large learning rate. However, this training accuracy increased slightly along with the decrease of the learning rate, and the whole history curve became flat. In contrast, the accuracy of the models using TL had a good initial value and continued to increase leading to convergence when the learning rate decreased, which was helpful in stabilizing the performance.

Table 6.2: Test accuracy results (%) for each case in the damage state task

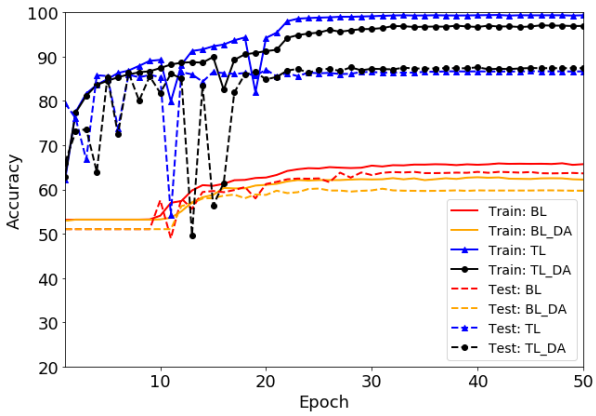
Model	**BL**	**BL-DA**	**TL**	**TL-DA**
VGG-16	64.0	60.7	87.3	87.6
VGG-19	62.7	60.7	87.3	87.4
ResNet-50	76.7	75.3	87.3	88.9

Task 3: Spalling condition (Material Loss)

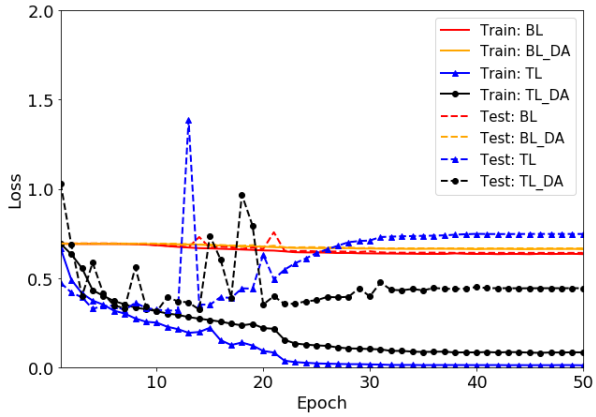
Results for test accuracy and training history of Task 3 (spalling condition) are shown in Table 6.3 and Fig. 6.3. Even though this task is also a binary classification like Task 2, its smaller amount of data (a total of 7,735 images, which was only 75% of that in Task 2) somewhat increased the difficulty. This explains the performance deterioration in the accuracy, where the best accuracy was 83.0% as obtained by ResNet-50 using both TL and DA. From the point of view of CV, the vision patterns of spalling are complex and irregular, which makes it difficult to identify, especially within the damaged component surface. Similar conclusions were obtained for Task 2, but the issues were more severe during the training of the model without using TL. As shown in Figs 6.3a to 6.3d, the loss and accuracy for training and testing of VGGNets became trapped. The models did not learn and, therefore,



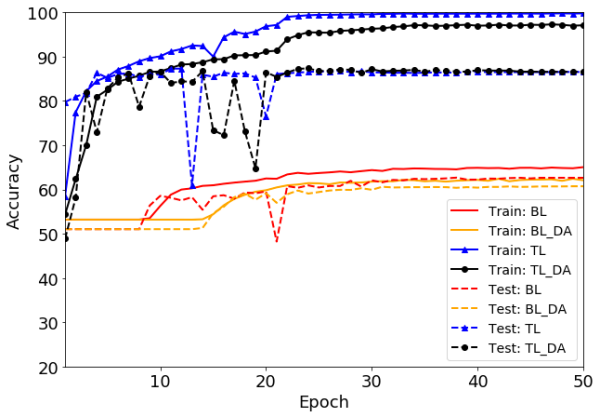
(a) Loss history of VGG-16



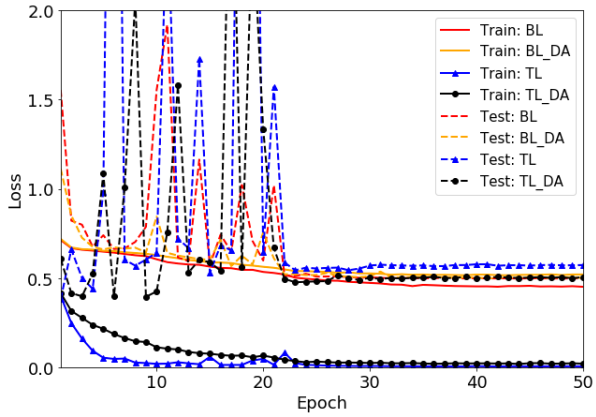
(b) Accuracy history of VGG-16



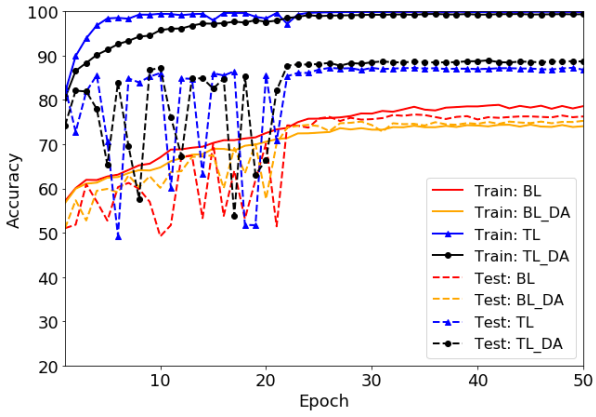
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.2: Training history of DL models for 50 epochs in the damage state task

could not update anything. Unlike VGGNets, ResNet-50 without TL was still trainable. However, after minor improvement, the model quickly converged to bad local minima. Similar to observations from Task 2, using TL improved the model performance significantly. On the other hand, using DA led to much less obvious improvements.

Table 6.3: Test accuracy results (%) for each case in the spalling condition task

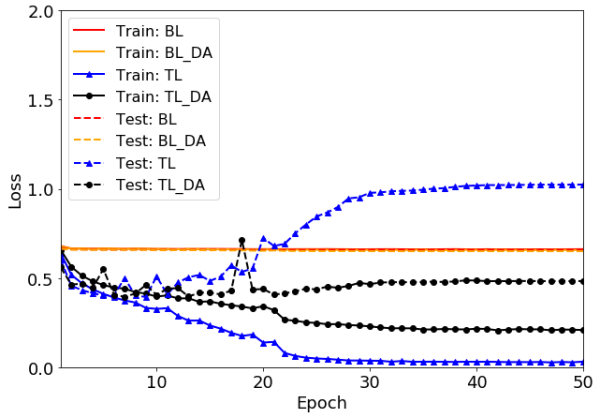
Model	BL	BL-DA	TL	TL-DA
VGG-16	63.0	63.0	81.6	81.5
VGG-19	63.0	63.0	81.2	81.8
ResNet-50	75.7	77.1	81.7	83.0

Task 4: Material type

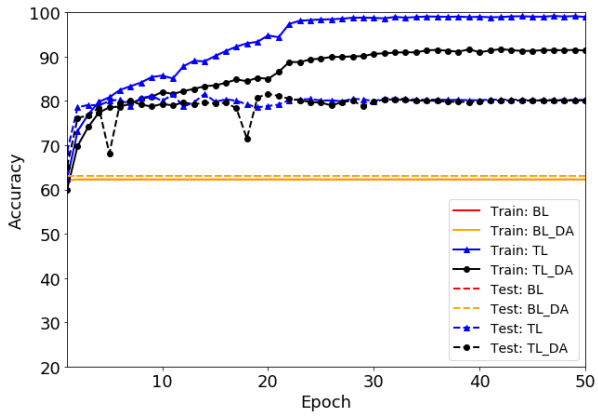
Results for test accuracy and training history of Task 4 (material type) are shown in Table 6.4 and Fig. 6.4. Unlike Tasks 2 and 3, the models were able to more easily identify the material type. The scope for material type observed from the surface is narrow, being defined into only two categories: "steel" and "other materials". Because there are sharp differences between these two vision patterns, i.e., texture, color, shape, etc., such distinct categories explain the high accuracy obtained for all cases, where the best results were achieved by ResNet-50 using both TL and DA. In this case, DA still did not contribute with any degree of significance toward improving the performance. The results were even worse when training from scratch, which is an observation similar to that obtained from Task 1. Another observation similar to Task 1 is that BL achieved high accuracy, whereas applying TL only achieved limited improvement of $\sim$3%. In general, the best performance at 98.5% is almost perfect and accordingly close to or even better than the expected human judgment by experts under some conditions. In the early training stage of ResNet, the models trained from scratch oscillated significantly no matter whether DA was adopted or not. This is also similar to that observed in Task 1. Overall, the DL approach was considered effective in distinguishing between steel and non-steel materials. It is expected to extend this benchmark by including finer classification categories, e.g., steel, masonry, wood, and concrete.

Table 6.4: Test accuracy results (%) for each case in the material type task

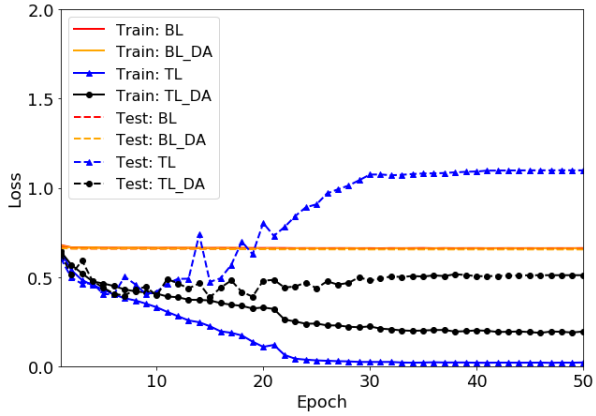
Model	BL	BL-DA	TL	TL-DA
VGG-16	93.9	90.8	97.0	97.0
VGG-19	94.4	90.3	96.5	96.9
ResNet-50	95.1	93.1	98.3	98.5



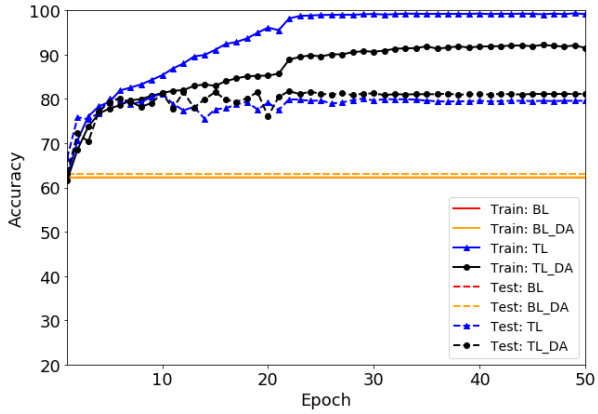
(a) Loss history of VGG-16



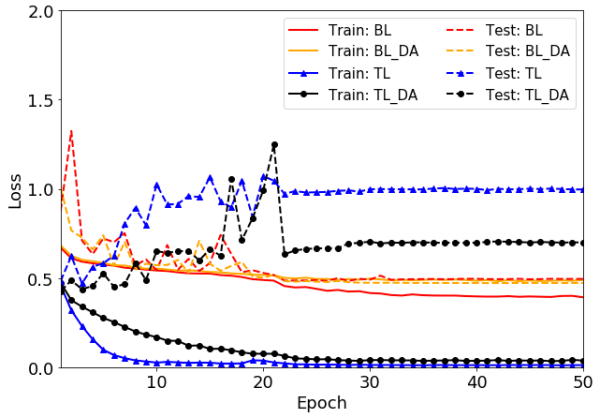
(b) Accuracy history of VGG-16



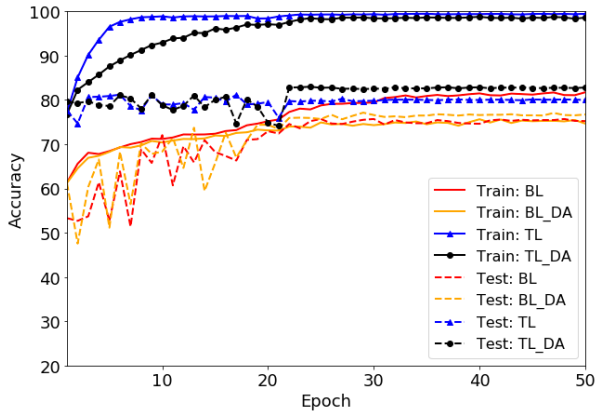
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19

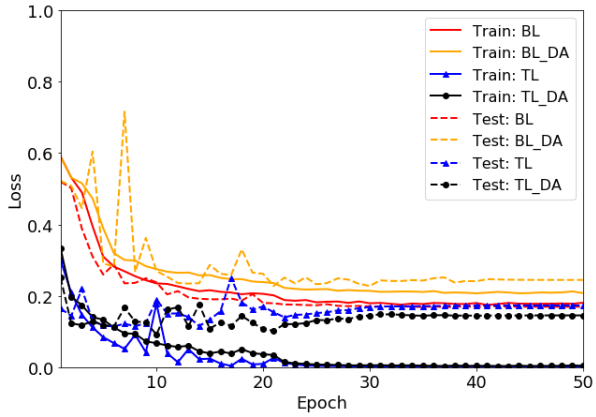


(e) Loss history of ResNet-50

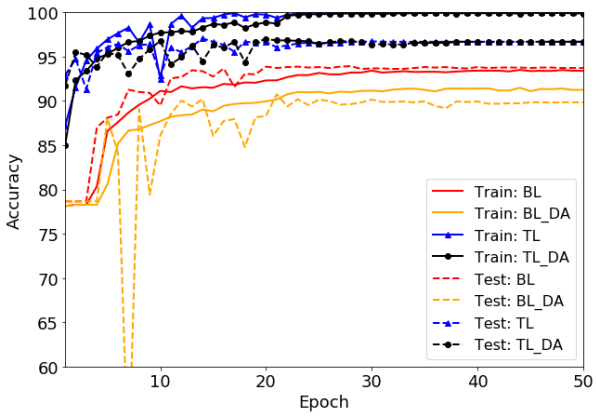


(f) Accuracy history of ResNet-50

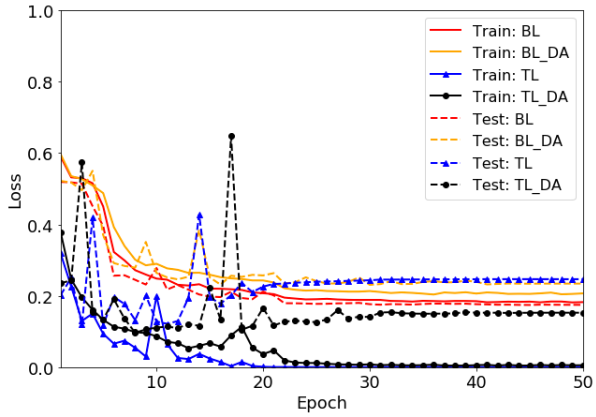
Figure 6.3: Training history of DL models for 50 epochs in the spalling condition task



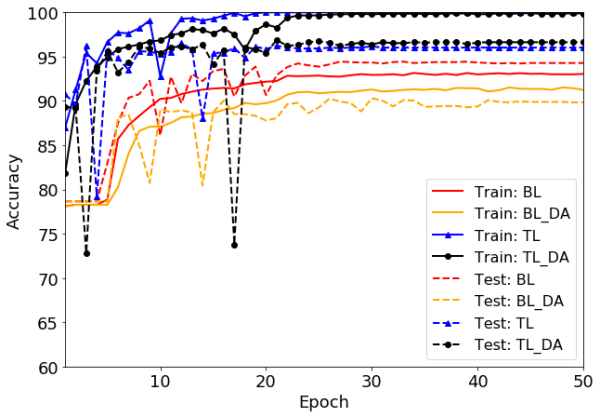
(a) Loss history of VGG-16



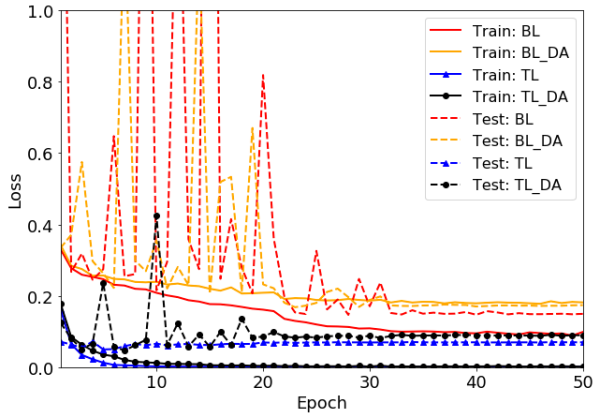
(b) Accuracy history of VGG-16



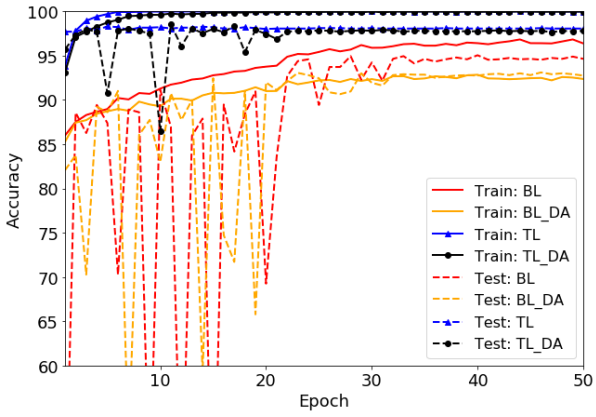
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.4: Training history of DL models for 50 epochs in the material type task

Task 5: Collapse mode

Results for test accuracy and training history of Task 5 (collapse mode) are shown in Table 6.5 and Fig. 6.5. Task 5 is a three-class classification, where the random guess is 33.3%. As mentioned above, this task is considered difficult due to very limited data (a total of 1,372), which is defined as an application of low-data regime. Thus, it was expected to obtain fewer promising results in terms of its baseline performance. Similar to issues noted in Task 2, VGGNets were untrainable when training from scratch, and the parameters did not change during the entire training history. The results for ResNet-50 were slightly better, but the final accuracy was still not acceptable. For models trained from scratch, DA did not perform satisfactorily. In contrast, using TL improved the model's performance significantly. As shown in Table 6.5, for the best model of VGG-16, VGG-19, and ResNet-50, the accuracy increased by ∼30%, 25%, and 13%, respectively, where DA contributed to the performance for VGG-19 but did not improve the performance of the other two models. Similar to the previous tasks, ResNet-50 using TL obtained the best test accuracy of 78.1%.

The number of data used in Task 5 (i.e., 1,372) is even less than that in ϕ-Net-2000 (i.e., 2,000) in the TL experiments in Chapter 5. Therefore, it is suggested that a similar partial network fine-tuning strategy as used in Chapter 5 can be considered in future studies to alleviate the over-fitting issues.

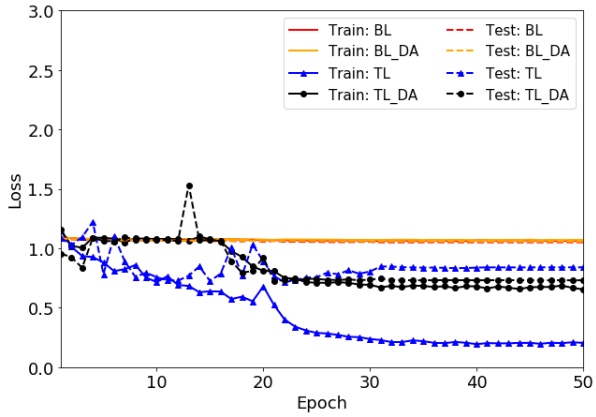
Table 6.5: Test accuracy results (%) for each case in the collapse mode task

Model	BL	BL-DA	TL	TL-DA
VGG-16	45.9	45.9	75.3	71.9
VGG-19	45.9	45.9	70.6	72.6
ResNet-50	65.1	61.6	78.1	74.7

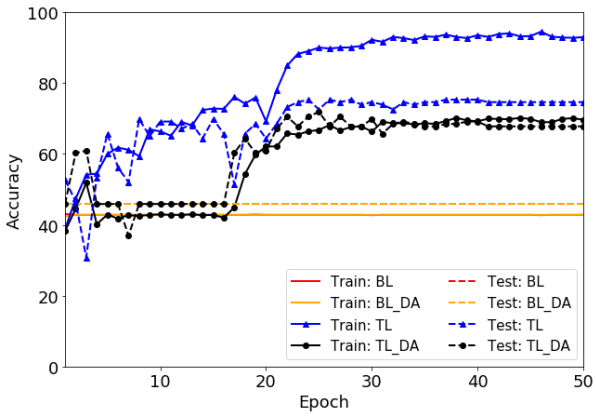
Task 6: Component type

Results for test accuracy and training history of Task 6 (component type) are shown in Table 6.6 and Fig. 6.6. Task 6 is a four-class classification, where the random guess is only 25%. The number of categories compared to previous tasks was expanded to four, which increased the difficulty for classification. Moreover, according to the hierarchy of the ϕ-Net dataset shown in Fig. 4.1, Task 6 is a leaf task of Task 1. Therefore, only images belonging to the object level flow through this task. This leads to a smaller image collection (a total of 5,334) for Task 6 compared to Task 1 (a total of 27,306) or Task 2 (a total of 13,271). These two issues underscore the major difficulty facing this component type task.

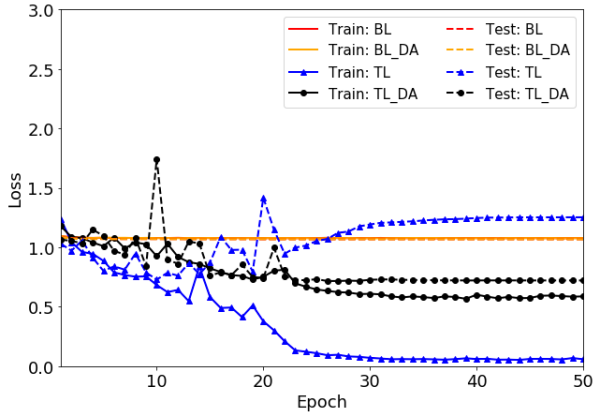
In general, the BL models obtained mediocre results but were better than the results obtained in Task 5. Again, DA did not perform satisfactorily. Similar to what was observed in the previous tasks, TL worked and provided about 10% improvement than BL. One interesting and valuable observation is that VGG-19 achieved the best accuracy (77.0%) as opposed to



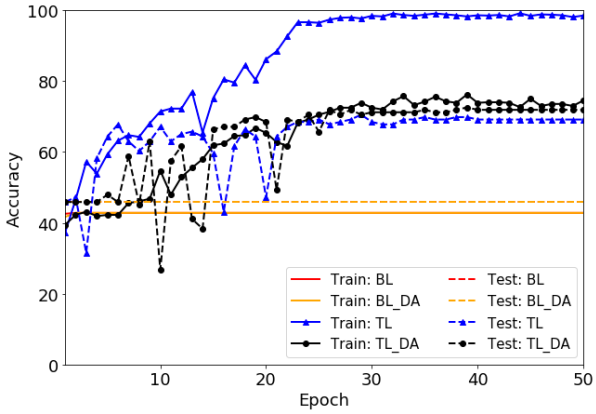
(a) Loss history of VGG-16



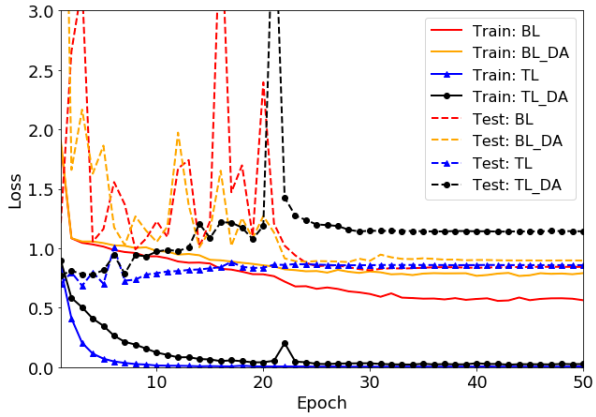
(b) Accuracy history of VGG-16



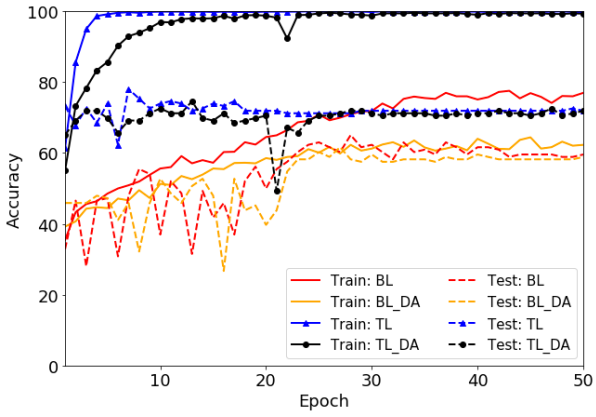
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.5: Training history of DL models for 50 epochs in the collapse mode task

ResNet-50. Such observations indicate the necessity of taking the DL model selection as a hyper-parameter where different models may be suitable for different application scenarios.

Table 6.6: Test accuracy results (%) for each case in the component type task

Model	BL	BL-DA	TL	TL-DA
VGG-16	64.8	64.8	75.1	76.9
VGG-19	65.1	65.1	77.0	76.9
ResNet-50	69.8	68.2	75.8	76.3

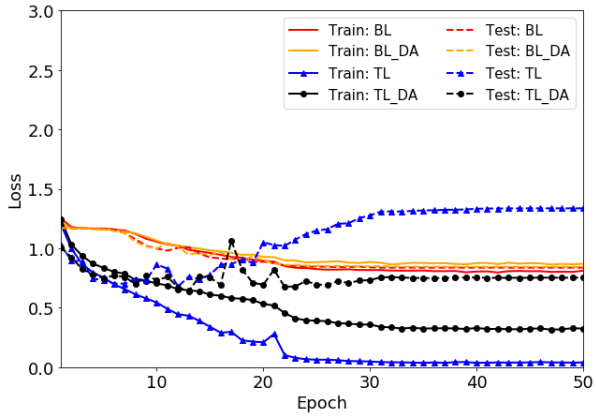
Task 7: Damage level

The results for test accuracy and training history of Task 7 (damage level) are shown in Table 6.7 and Fig. 6.7. Task 7 is a four-class classification. Compared to Task 6, it is more abstract and has semantic meaning, which significantly increases the difficulty. Here, the so-called abstract and semantic meaning means that the categories cannot be easily identified by low-level vision features such as edge, color, etc., as is the case of Task 6. The classification herein needed to be elevated to high-dimensional spaces to identify some key features linked to various damage patterns semantically. For example, the orientation and boundary of the component can lead to the specific component type. As pointed out in the task and label definitions of Task 7 in Chapter 4, damage level relates more to the severity and development of damage recognized by humans, which is often subjective and not easily quantifiable. Thus, it requires the DL model to find features and patterns in abstract and high-dimensional spaces related to semantic meaning of "minor", "moderate", or "heavy", which is more difficult compared to the previous tasks. As is the case in Task 6, the total amount of images available for this task was only 4,636, which is considered inadequate.

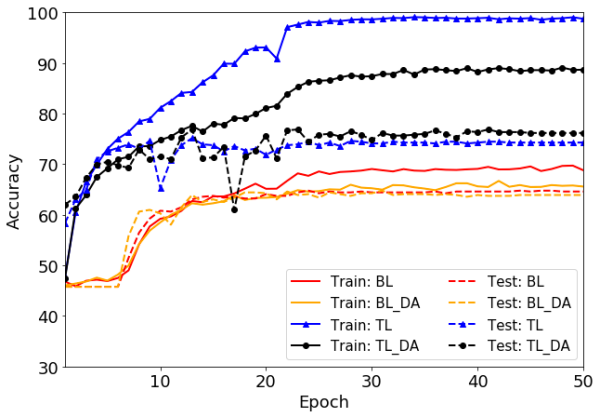
From Table 6.7, the performance of BL was consistent with the concerns that VGGNets were untrainable no matter whether DA was adopted or not. The test accuracy was unacceptable and was only $\sim$41%. The observation in BL was similar to that found in Task 6, but there were some differences. The ResNet-50 trained from scratch was capable of obtaining a much better BL performance compared to VGGNets, and using DA further improved the performance by $\sim$4%. Due to the relatively poor performance in BL, models with TL were considered significantly improved from those trained from scratch, with improvements of $\sim$30%, 23%, and 18% for VGG-16, VGG-19, and ResNet-50, respectively. ResNet-50 still performed the best with 74.5% test accuracy, which is acceptable for this task using the current small dataset.

Task 8: Damage type

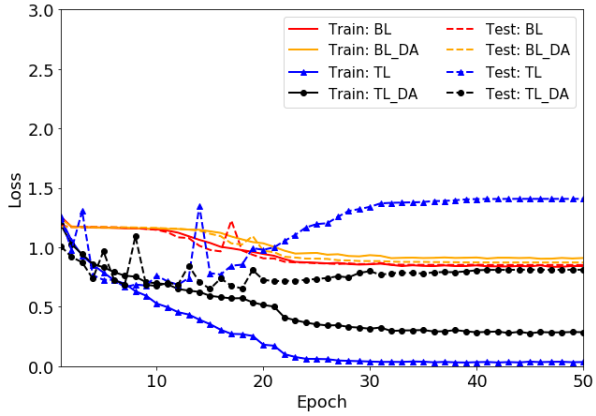
Results for test accuracy and training history of Task 8 (damage type) are shown in Table 6.8 and Fig. 6.8. Similar to Task 7, Task 8 is also thought of as abstract with various



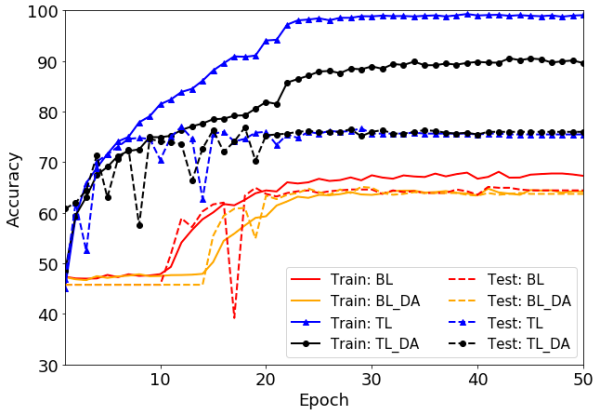
(a) Loss history of VGG-16



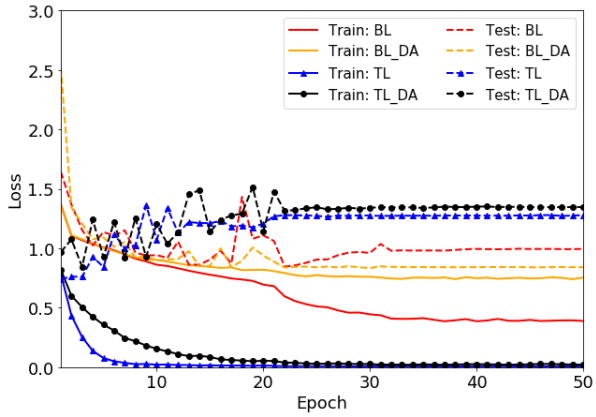
(b) Accuracy history of VGG-16



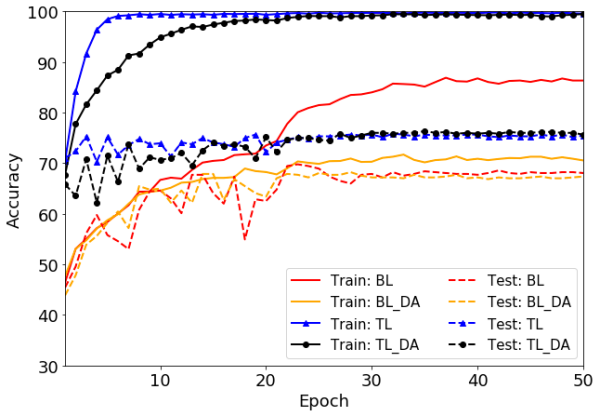
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19

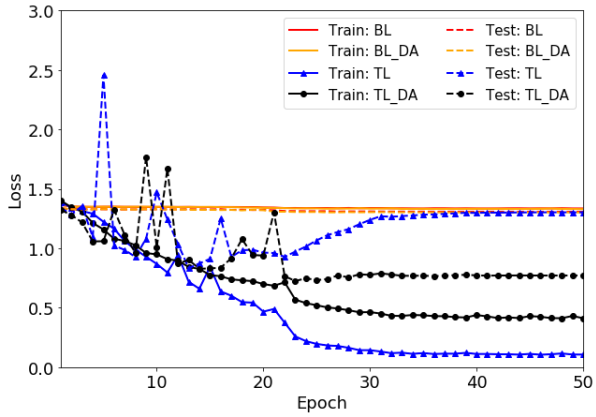


(e) Loss history of ResNet-50

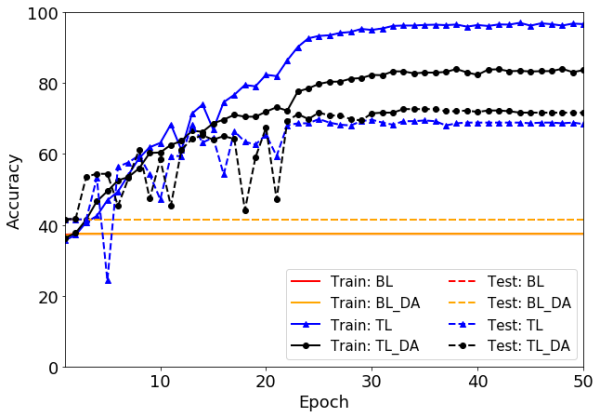


(f) Accuracy history of ResNet-50

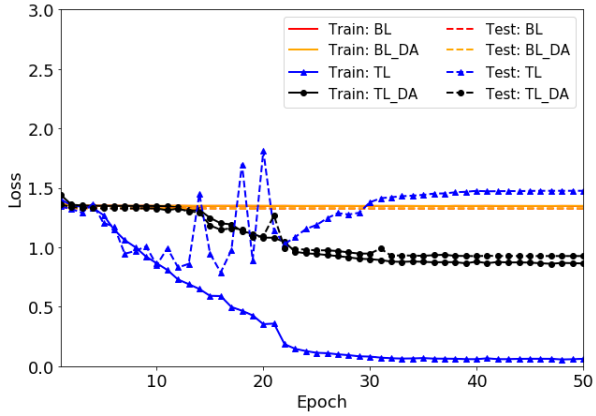
Figure 6.6: Training history of DL models for 50 epochs in the component type task



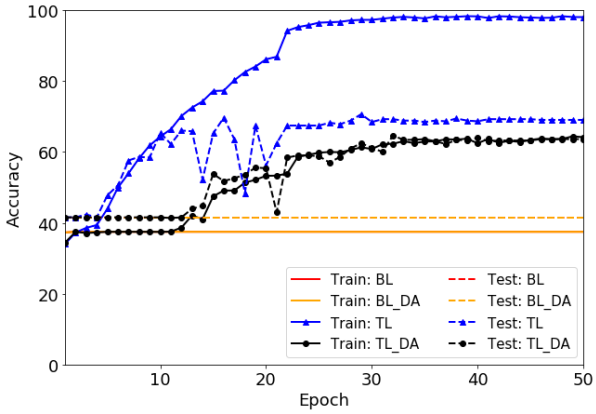
(a) Loss history of VGG-16



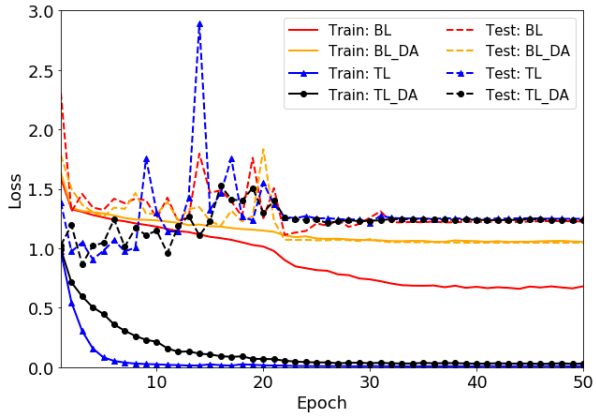
(b) Accuracy history of VGG-16



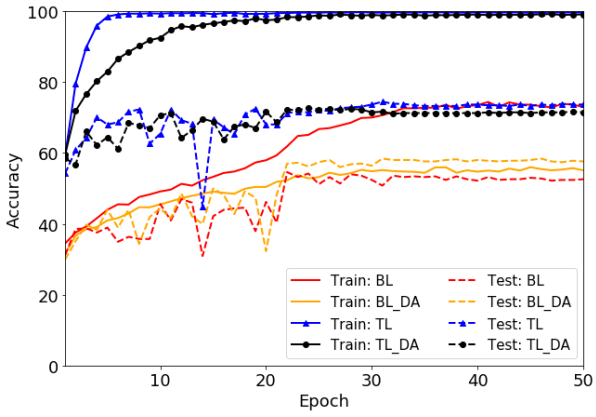
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.7: Training history of DL models for 50 epochs in the damage level task

Table 6.7: Test accuracy results (%) for each case in the damage level task

Model	BL	BL-DA	TL	TL-DA
VGG-16	41.6	41.6	69.9	72.7
VGG-19	41.6	41.6	70.7	64.7
ResNet-50	54.6	58.4	74.5	72.7

vision patterns corresponding to specific types mentioned in the task definition in Chapter 4. From the perspective of the relative lack of data, a total of available 4,585 images is not considered representative enough for training purpose. Similar issues occurred herein to those of BL models for Task 7, i.e., VGGNets proved to be untrainable, and ResNet-50 achieved relatively low accuracy. In contrast, using TL achieved significant improvement in this task, and the best accuracy of VGG-16, VGG-19, and ResNet-50 increased by ∼27%, 29%, and 15%, respectively. Among the above three models, VGG-19 performed the best (as was also found for Task 6). It is noted that the performance of the best ResNet-50 model was even worse than the performance for the VGG-16 model. This underscores the importance of selecting the appropriate model for future applications instead of relying on only a single model.

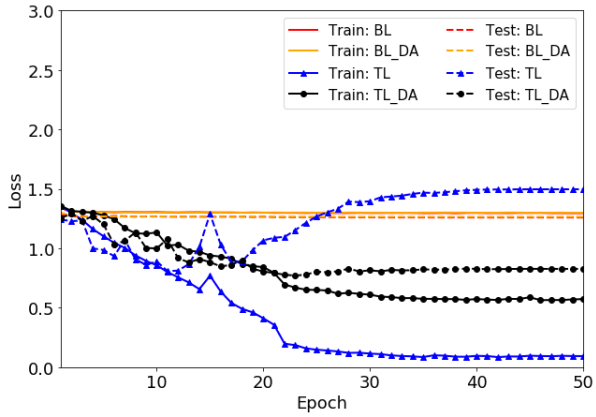
Table 6.8: Test accuracy results (%) for each case in the damage type task

Model	BL	BL-DA	TL	TL-DA
VGG-16	43.7	43.7	70.7	70.5
VGG-19	43.7	43.7	71.1	72.4
ResNet-50	54.3	56.1	68.9	68.1

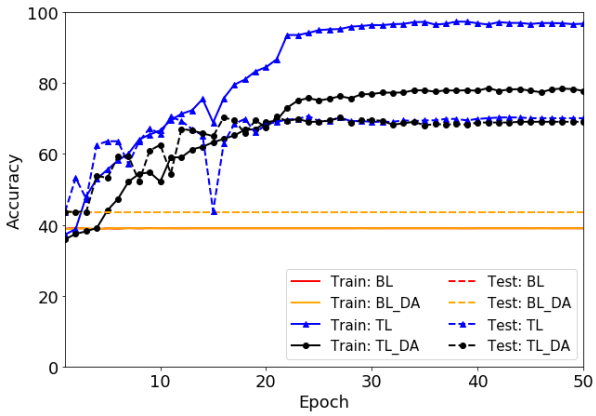
6.4 Summary

The above sections presented the reference results including history plots for all cases in the identified eight tasks, and Table 6.9 lists the best reference values. Since the ultimate goal of this study is to adopt DL for automated detection, the accuracy should be within a reasonable range. For the easier tasks (Tasks 1 to 4), the test accuracy appears very promising, especially for classification between steel and non-steel materials. For the harder tasks (Tasks 5 to 8), although the best results are still reasonable and acceptable, they are expected to be improved with more data, more sophisticated DL models, training approaches, and tuned hyper-parameters.

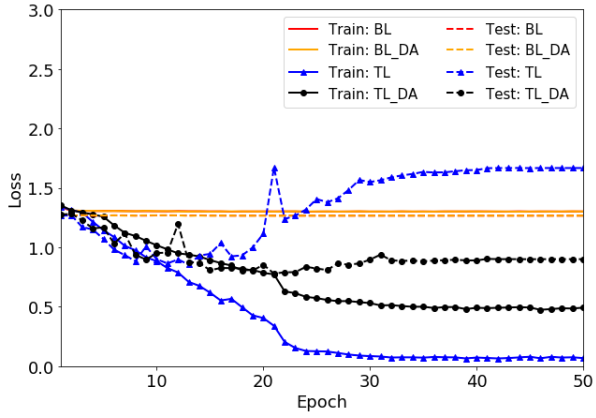
In a comparison of the DL models, ResNet-50 appeared to be more generalized because it achieved the highest test accuracy in most cases (except in Tasks 6 and 8). Thus, to achieve the state-of-the-art performance, it is suggested to perform model selection among multiple DL models. From the perspective of the training approach, TL worked well in all eight tasks,



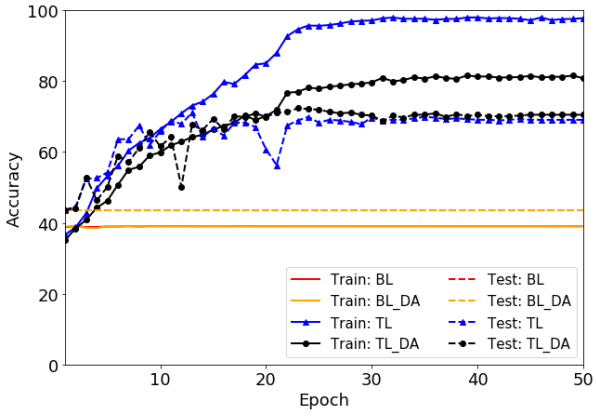
(a) Loss history of VGG-16



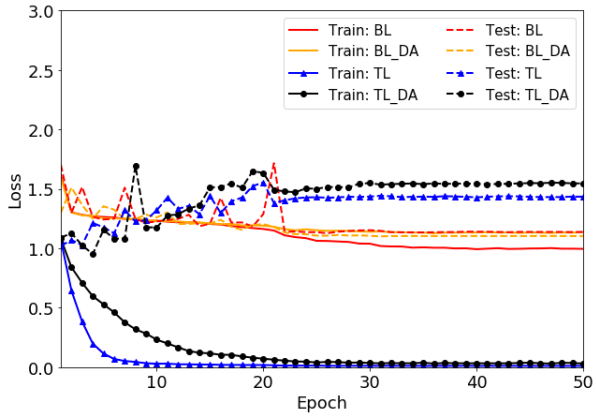
(b) Accuracy history of VGG-16



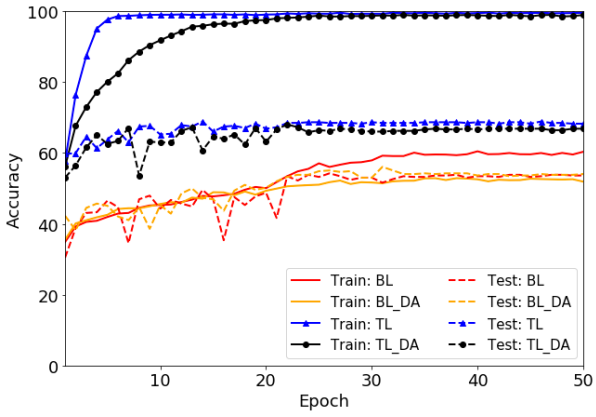
(c) Loss history of VGG-19



(d) Accuracy history of VGG-19



(e) Loss history of ResNet-50



(f) Accuracy history of ResNet-50

Figure 6.8: Training history of DL models for 50 epochs in the damage type task

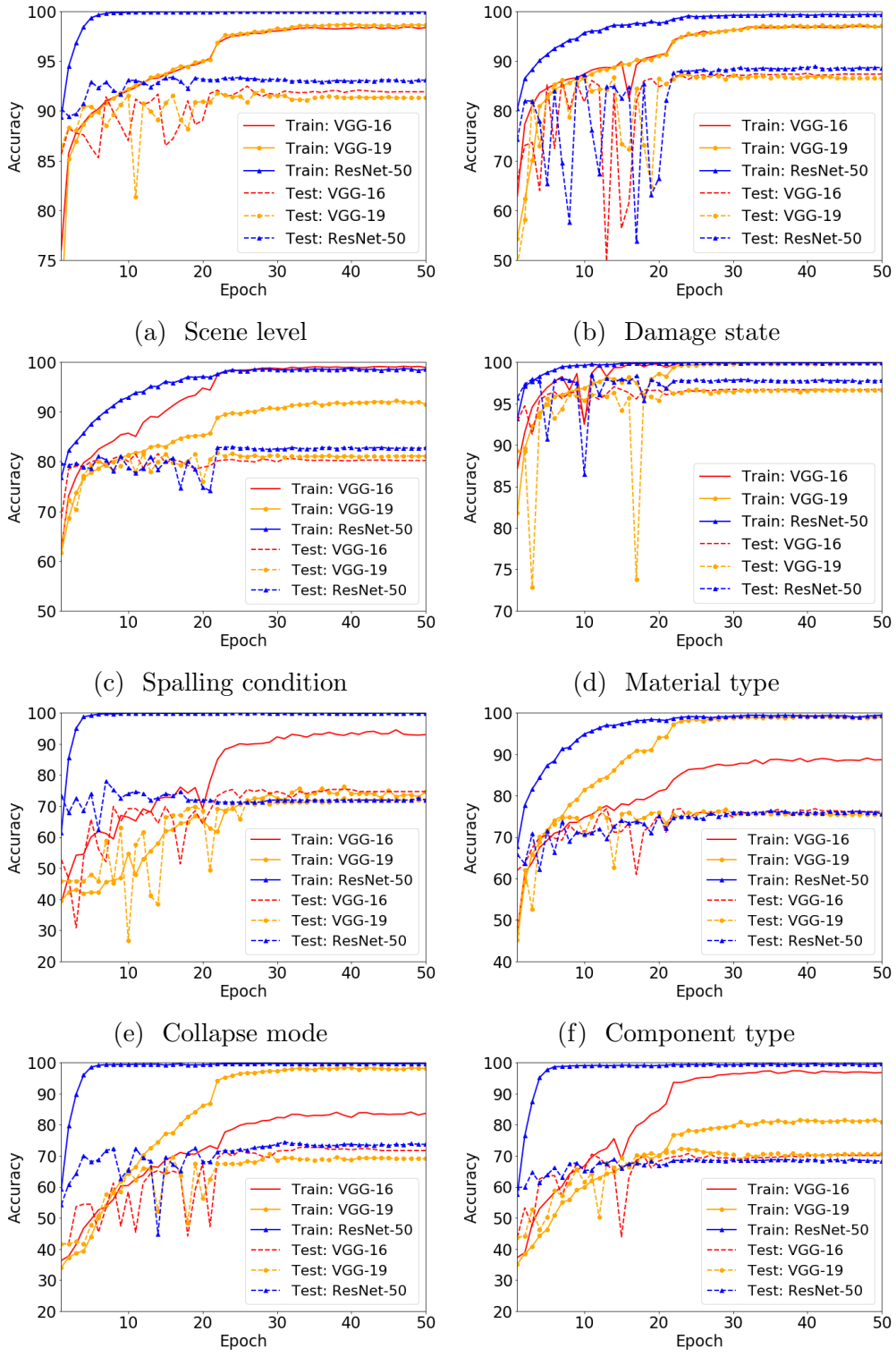
with significant improvement in many cases, and could be seen as the necessary training strategy for the current scale of the dataset. It is inferred that the ImageNet pre-trained models already have some sense on features of basic and natural objects from the source domain. Thus, transferring source-domain knowledge to the target domain of structural images may provide a better understanding for the deep CNN models towards some general structural vision patterns, e.g., object boundary, material texture, etc. Moreover, fine-tuning from the ImageNet pre-trained models obtains better parameter initialization than directly training models from scratch where parameters are initialized randomly or from certain distributions, e.g., Gaussian. These findings are more evident in the harder tasks with limited amount of data or abstract semantic meanings, e.g., TL achieved 10$\sim$30% enhancements over the BL model cases. Compared to TL technique, DA did not significantly improve the accuracy of the test and in some cases performed poorly compared to other approaches. This can be attributed to two major issues: (1) randomness and uncertainty due to only having one run for each case; and (2) different DA settings, e.g., range of rotation angle, translation, etc., which may lead to different performance.

Table 6.9: Summary of the best test accuracy performance (%) in all tasks

Task	Best result	Model	Approach	Over BL	Over BL-DA	Over TL	Over TL-DA
Scene level	93.4	ResNet-50	TL	+5.7	+5.4	-	+0.4
Damage state	88.9	ResNet-50	TL-DA	+12.2	+13.7	+1.7	-
Spalling condition	83.0	ResNet-50	TL-DA	+7.3	+5.9	+1.8	-
Material type	98.5	ResNet-50	TL-DA	+3.5	+5.4	+0.2	-
Collapse mode	78.1	ResNet-50	TL	+13.0	+16.3	-	+3.4
Component type	77.0	VGG-19	TL	+11.9	+11.9	-	+0.2
Damage level	74.5	ResNet-50	TL	+19.9	+16.1	-	+1.8
Damage type	72.4	VGG-19	TL-DA	+28.7	+28.7	+1.2	-

More information about a model's performance can be extracted from the training and test history plots. Thus, the best results for VGG-16, VGG-19, and ResNet-50 are plotted in one figure for each task; see Figs 6.9a to 6.9h. In all benchmark tasks for all models, the training accuracy is high and close to 100%. However, the gap between training accuracy and that of the test is evident and it varies from one task to another. This is an indicator of over-fitting and fully training, i.e., the models contained enough parameters and complexity to learn the task well. In ML and DL, usually applying regularization, i.e., dropout (applied herein but the dropout rate can be adjusted), batch normalization [47], etc., may help to alleviate over-fitting. This issue is worthy of future studies. From the perspective of training stability, DL models are less stable in the early training stage when the learning rate is large. During this period, the parameters are explored and updated more with SGD, and the history curves are more oscillating. With a scheduled decayed learning rate (especially after 20 epochs), the training curves become flat and smooth, indicating that the model performance is becoming stable.

Computational efficiency is another important metric in DL applications. Table 6.10 lists average computational times per training epoch for each model under different training



(a) Scene level (b) Damage state

(c) Spalling condition (d) Material type

(e) Collapse mode (f) Component type

(g) Damage level (h) Damage type

Figure 6.9: Accuracy history of the best models for 50 epochs in all tasks

approaches, thus providing a reference for future studies. Without DA, ResNet-50 costs the least computational time, VGG-16 costs the second least, and VGG-19 is the most expensive one as it contains more parameters than the other two networks. There are roughly 21 million, 26 million, and 25 million parameters in VGG-16, VGG-19, and ResNet-50, respectively. Even though ResNet-50 is deeper than VGGNets, using a smaller filter size (1×1 rather than the 3×3 applied in VGGNets) and shortcut connections that make gradient flow faster accelerate the training speed for the ResNet-50. With DA, all models have increasing time costs and similar training time per epoch due to the extra cost of performing transformation on raw images. As for TL, it does not affect the average training time, because it neither introduces new parameters nor performs image-processing operations. Thus, using TL is proved to be a more efficient and economical method in the experiments reported herein.

In summary, the reference results and history plots for all cases and tasks are reported, and then compared with the performance of the best DL models. The influences of DA and TL are analyzed, and the computational time cost for each task is assessed. For benchmarking purposes, a comparison with the same settings is conducted for all tasks. It is noted that although the results shown herein are not the state-of-the-art, they provide a reference for future DL applications. In order to obtain better results, tuning hyper-parameters (e.g., learning rate, dropout rate, the number of neurons in the fc-layers, etc.), developing new models, and training approaches are proposed for future studies. It is recommended to make full use of the ϕ-Net dataset to enlarge the user's own dataset, to quantify finer categories, or to relabel images for new vision tasks.

Table 6.10: Average computational time (seconds) per training epoch in all tasks

Task	**Model**	**BL**	**BL-DA**	**TL**	**TL-DA**
Scene level	VGG-16	145	178	146	187
	VGG-19	169	185	169	188
	ResNet-50	131	183	123	190
Damage state	VGG-16	71	79	71	81
	VGG-19	82	83	82	83
	ResNet-50	64	83	63	83
Spalling condition	VGG-16	42	46	42	46
	VGG-19	48	49	48	49
	ResNet-50	37	47	37	47
Material type	VGG-16	52	67	52	67
	VGG-19	59	69	59	69
	ResNet-50	46	68	46	68
Collapse mode	VGG-16	8	8	8	8
	VGG-19	9	9	9	9
	ResNet-50	7	8	7	8
Component type	VGG-16	28	30	28	30
	VGG-19	33	33	33	33
	ResNet-50	26	30	26	31
Damage level	VGG-16	25	26	25	26
	VGG-19	29	29	29	29
	ResNet-50	22	27	22	27
Damage type	VGG-16	25	25	25	25
	VGG-19	28	29	28	29
	ResNet-50	22	27	22	27

Chapter 7

Applications & Extensions of the ϕ-Net

Chapter 6 presented a comprehensive investigation on the built ϕ-Net using DL methods. In this chapter, two applications and extensions with respect to the ϕ-Net are presented and discussed, namely image-based post-disaster damage assessment using ϕ-Net pre-trained model and PEER Hub ImageNet (ϕ-Net) Challenge.

7.1 Post-disaster Damage Assessment

Structural ImageNet Model (SIM)

Inspired by pre-training deep CNN [24], e.g., VGGNet [113], InceptionNet [122], and ResNet [42], on the ImageNet [24] and then adapting them to new target-domain tasks, the concept of *Structural ImageNet Model* (*SIM*) is proposed in this chapter. In the SIM, if a DL model is pre-trained on ϕ-Net can obtain acceptable performance, it is thought of as a good source model to be transferred to other structural vision tasks by further fine-tuning, or can be directly applied into practical scenarios, e.g., post-disaster damage assessment.

From the best results in ϕ-Net benchmark experiments (Table 6.9), several best models for easy tasks (i.e., 1 to 4) achieved a promising accuracy. Thus, they are expected to be directly used in real scenarios. For example, the best model for Task 1 (scene level) has 93.4% accuracy in identifying the scene level of the structural images and that for Task 4 (material type) has 98.5% accuracy in classifying materials between steel and non-steel. These two models can be directly applied to image collection and pre-processing procedures in practice to mitigate laborious human efforts in organizing different scale and material type data. Meanwhile, the best models in hard tasks (i.e., 5 to 8) still achieved acceptable results, so they can also be treated as source models for further fine-tuning to more complex tasks with larger dataset and finer class categories. For example, in Task 6 (component type) due to the lack of variant structural images, the component type is limited to four classes,

	# 1	# 2	# 3	# 4	# 5
Real image					
Scene Level	P: 0.0% O: 0.0% ***S: 100.0%***	P: 0.0% O: 0.6% ***S: 99.4%***	P: 0.0% ***O: 100.0%*** S: 0.0%	P: 0.6% ***O: 99.4%*** S: 0.0%	P: 0.0% ***O: 99.9%*** S: 0.1%
Damage State	***D: 99.3%*** UD: 0.7%	***D: 99.8%*** UD: 0.2%	***D: 100.0%*** UD: 0.0%	***D: 99.7%*** UD: 0.3%	***D: 99.5%*** UD: 0.5%
Spalling condition	-	-	NSP: 0.9% ***SP: 99.1%***	NSP: 0.1% ***SP: 99.9%***	NSP: 0.1% ***SP: 99.9%***
Collapse Mode	NC: 6.3% PC: 14.3% ***GC: 79.4%***	NC: 0.1% PC: 5.6% ***GC: 94.3%***	-	-	-
Component Type	-	-	Beam: 0.3% Col: 45.2% *Wall: 49.7%* Else: 4.8%	Beam: 0.3% Col: 12.2% ***Wall: 79.9%*** Else: 7.6%	Beam: 2.7% ***Col: 57.9%*** Wall: 14.1% Else: 25.3%

Figure 7.1: Results predicted by the SIM for the 1999 Chi-Chi earthquake images

including the general label "others." Moreover, if users collected more labeled images about "others," it can be split into finer classes, e.g., brace, joint, or non-structural components.

Application of SIM for the 1999 Chi-Chi earthquake

In this section, an application of the ϕ-Net with SIM is presented for rapid assessment of structural images from the 1999 Chi-Chi earthquake, Taiwan, where unlabeled images were collected from Google Images[1]. The detection procedure is straightforward via feeding the collected images into the trained SIM corresponding to the task of interest, then the model outputs the predicted label and the probabilities of each possible label choice. This operation is repeated for multiple tasks, then a comprehensive set of results is obtained for decision-making purposes. Based on the fact that most collected images belong to the object and the scene levels, the application is simplified to focus on the following five tasks among the original ϕ-Net benchmark tasks: (1) scene level, (2) damage state, (3) spalling condition, (4) collapse mode, and (5) component type. If the image corresponds to the object level, it is explored for spalling condition and component type. On the other hand, if the image corresponds to the structural level, it is explored for the collapse mode task.

Fig. 7.1 lists the attributes of five sample images from the 1999 Chi-Chi earthquake, where the SIMs are the best models in the corresponding tasks from Table 6.9. In Fig. 7.1, the predicted label for each task is selected as the one with the highest probability and shown in

[1] `https://images.google.com`.

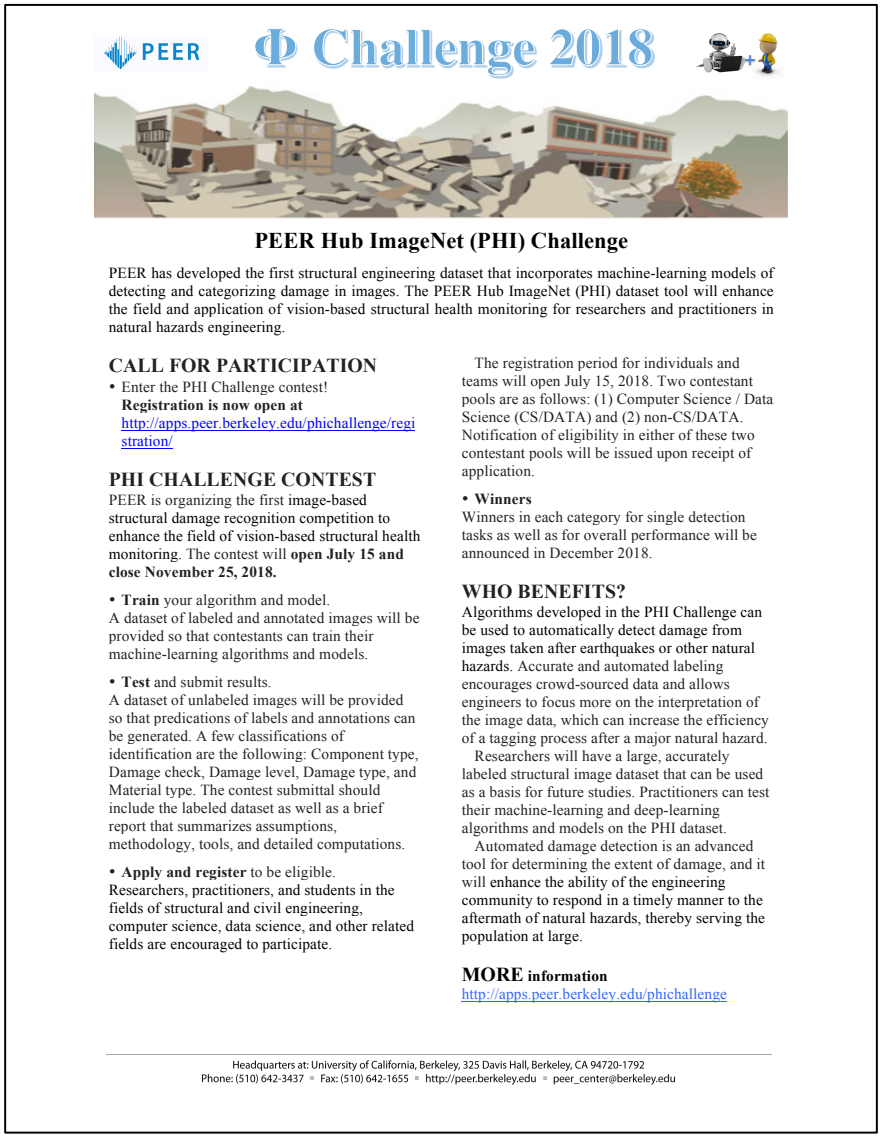
bold italic font. Moreover, the bold italic values imply that the predictions are consistent with the ground truth but the single underlined value represents a wrong prediction. Generally speaking, SIMs achieved satisfactory results for these newly seen images indicating good model generalization. Especially in the easy tasks (scene level, damage state, and spalling condition), the predicted probability for the correct label is almost 100% showing the high confidence of the SIM towards such tasks. The one incorrect example corresponds to the SIM making a mistake by predicting the component type of image #3 as a wall instead of a column. From the perspective of the vision pattern, this column is a boundary column and on the right side of a wall. Thus, it is highly possible that the texture and boundary of the wall have confused the SIM. Moreover, labels "column" and "wall" have very closely predicted probabilities, 45.2% and 49.7%, respectively, reflecting the low confidence of the model under the ambiguity of the input image #3. On the contrary, images #4 and #5 with clear vision patterns lead to more accurate results and higher confidence. As for the time cost, the average computational time for a single image in each task is within 0.1 second (i.e., considered as rapid "real time" or at least "near real time" assessment) including the numerical encoding of the image and performing the classification.

In summary, the SIMs can achieve the desirable performance in rapid assessment even though only benchmarking (not state-of-the-art) models were adopted to develop these SIMs. The multiple attributes extracted by the SIMs can provide information for decision-making purposes, and the detection accuracy can be further improved with the use of better trained DL models in the future. Therefore, this application indicates the practical uses and benefits of the ϕ-Net and the well-trained SIMs for vision-based SHM.

7.2 2018 ϕ-Net Challenge

Chapters 4 to 6 described the definition, development procedure, preliminary investigation, and benchmark experiments of the ϕ-Net. Before finalizing and open-sourcing the benchmark dataset and results, during the Fall of 2018, the Pacific Earthquake Engineering Research (PEER) Center released a partial dataset of the ϕ-Net dataset and held a worldwide challenge to call for interested researchers to participate. The results of the challenge provided a preliminary reference for the establishment of the final benchmark dataset. The challenge also served as a means to alert the engineering community and encourage researchers in civil engineering to interact with other disciplines towards transdisciplinary research activities[2], and to learn and apply the state-of-the-art approaches in CV for applications in civil (particularly, structural) engineering.

[2]Efforts conducted by investigators from different disciplines working jointly to create new conceptual, theoretical, methodological, and translational innovations that integrate and move beyond discipline-specific approaches to address a common problem [49].



PEER

Φ Challenge 2018

PEER Hub ImageNet (PHI) Challenge

PEER has developed the first structural engineering dataset that incorporates machine-learning models of detecting and categorizing damage in images. The PEER Hub ImageNet (PHI) dataset tool will enhance the field and application of vision-based structural health monitoring for researchers and practitioners in natural hazards engineering.

CALL FOR PARTICIPATION

- Enter the PHI Challenge contest!
  Registration is now open at
  http://apps.peer.berkeley.edu/phichallenge/registration/

PHI CHALLENGE CONTEST

PEER is organizing the first image-based structural damage recognition competition to enhance the field of vision-based structural health monitoring. The contest will **open July 15 and close November 25, 2018.**

- **Train** your algorithm and model.
  A dataset of labeled and annotated images will be provided so that contestants can train their machine-learning algorithms and models.
- **Test** and submit results.
  A dataset of unlabeled images will be provided so that predications of labels and annotations can be generated. A few classifications of identification are the following: Component type, Damage check, Damage level, Damage type, and Material type. The contest submittal should include the labeled dataset as well as a brief report that summarizes assumptions, methodology, tools, and detailed computations.
- **Apply and register** to be eligible.
  Researchers, practitioners, and students in the fields of structural and civil engineering, computer science, data science, and other related fields are encouraged to participate.

The registration period for individuals and teams will open July 15, 2018. Two contestant pools are as follows: (1) Computer Science / Data Science (CS/DATA) and (2) non-CS/DATA. Notification of eligibility in either of these two contestant pools will be issued upon receipt of application.

- **Winners**
  Winners in each category for single detection tasks as well as for overall performance will be announced in December 2018.

WHO BENEFITS?

Algorithms developed in the PHI Challenge can be used to automatically detect damage from images taken after earthquakes or other natural hazards. Accurate and automated labeling encourages crowd-sourced data and allows engineers to focus more on the interpretation of the image data, which can increase the efficiency of a tagging process after a major natural hazard.

Researchers will have a large, accurately labeled structural image dataset that can be used as a basis for future studies. Practitioners can test their machine-learning and deep-learning algorithms and models on the PHI dataset.

Automated damage detection is an advanced tool for determining the extent of damage, and it will enhance the ability of the engineering community to respond in a timely manner to the aftermath of natural hazards, thereby serving the population at large.

MORE information

http://apps.peer.berkeley.edu/phichallenge

Headquarters at: University of California, Berkeley, 325 Davis Hall, Berkeley, CA 94720-1792
Phone: (510) 642-3437 ▫ Fax: (510) 642-1655 ▫ http://peer.berkeley.edu ▫ peer_center@berkeley.edu

Figure 7.2: Flyer for the 2018 ϕ-Net Challenge

Motivation of the ϕ-Net Challenge

Inspired by several famous CV competitions in the computer science field, such as the ImageNet Challenge (ILSVRC)[3] [24], PASCAL VOC Challenge[4], and COCO Challenge[5], during the Fall of 2018, PEER organized the first image-based structural damage identification competition, namely the ϕ-Net Challenge. Fig. 7.2 shows the advertising flyer for this challenge.

[3] `http://image-net.org/challenges/LSVRC/`.
[4] `http://host.robots.ox.ac.uk/pascal/VOC/`.
[5] `http://cocodataset.org/#home`.

In this competition, PEER provided eight image datasets that corresponded to the tasks introduced in Section 4.1. Each team received training and test sets for these tasks. It is noted that since the ϕ-Net Challenge acted as a pre-event for the open-sourcing of the ϕ-Net dataset, the datasets released in the challenge were different from the finalized benchmark datasets summarized in Sections 4.2 & 4.3. Based on these well-labeled images, each team was expected to develop and use algorithms to train their recognition models and then predict labels for the unlabeled test datasets. The labels predicted for the test datasets would then be compared against the ground truth. Teams with the highest accuracy were declared the winners of the competition. The challenge started on 23 August 2019, at 00:00 am UTC and was closed on 25 November 2019 at 11:59 pm UTC. For more details about the ϕ-Net Challenge, refer to the official website[6].

The ϕ-Net dataset and challenge effort represent an essential part of the PEER's strategic plan to equip the earthquake engineering community with the tools of the digital revolution era, including ML, DL, AI, high-performance computing (HPC), and emerging technology. The main objective of the challenge was to fully engage the earthquake engineering community (or more broadly the natural hazards communities) in all stages of the competition, including preparation for the computation, execution of the computation, and processing of the results. In the competition preparation stage, members of the community contributed significantly by uploading images that could be used in the competition and by labeling these images through the developed tool mentioned in Section 4.2. PEER welcomes contributions of newly labeled images to further expand the current ϕ-Net dataset.

In summary, the goal of the ϕ-Net Challenge was to evaluate algorithms for structural image classification using large-scale datasets based on service conditions, past reconnaissance efforts, and laboratory experiments for condition assessments following extreme events. The state-of-the-art algorithms tested in the challenge are expected to enhance the accuracy and the generalization of the vision-based approaches for SHM, and to contribute to the construction of a large structural image dataset used to solve societal-scale problems of SHM and to perform assessments of the built environment.

Participation in the ϕ-Net Challenge

Team setup

A team consisted of either an individual or a maximum of four members. The team setting rules encouraged communication and collaboration between different disciplines. In the 2018 ϕ-Net Challenge, PEER received a total of sixty-eight team applications from different countries and regions from around the world. The geographic distribution of the registered teams is shown in Fig. 7.3, where over half of the teams were from the United States. As expected, many other disciplines besides structural engineering were actively engaged in the challenge, including mechanical engineering, transportation engineering, environmental engineering, computer science, and statistics. The challenge considered the background

[6] `https://apps.peer.berkeley.edu/phichallenge/`.



Figure 7.3: Registered team distribution of the 2018 ϕ-Net Challenge

advantage of those participants from computer science and data science in ML/DL, and two different pools were created to address this issue: computer science/data science (CS/DATA) and engineering (Non-CS/DATA). Each team was assigned to the pool corresponding to the team members' educational fields as declared in the team application. If at least one team member was majoring or had a major in computer science/data science, that team was assigned to the pool of CS/DATA and vice versa.

Access to the challenge

The 2018 ϕ-Net Challenge followed the current trend of deploying data science competitions on Kaggle[7], which is an online community of data scientists and machine learners. Kaggle provides a flexible platform to host online challenges where the host can release the dataset and launch the challenge in one of multiple ways, including: (1) public challenges, (2) invited challenges, or (3) private challenges. In the 2018 ϕ-Net Challenge, eight challenge webpages were established via Kaggle as invited challenges, and only the teams who applied could receive the links to access the data and later submit their results, refer to Fig. 7.4. The Kaggle webpages for the eight tasks mentioned above are as follows:

- Task 1 (Scene level): `https://www.kaggle.com/c/phi2018task1`
- Task 2 (Damage state): `https://www.kaggle.com/c/phi2018task2`

[7] `https://www.kaggle.com/`.

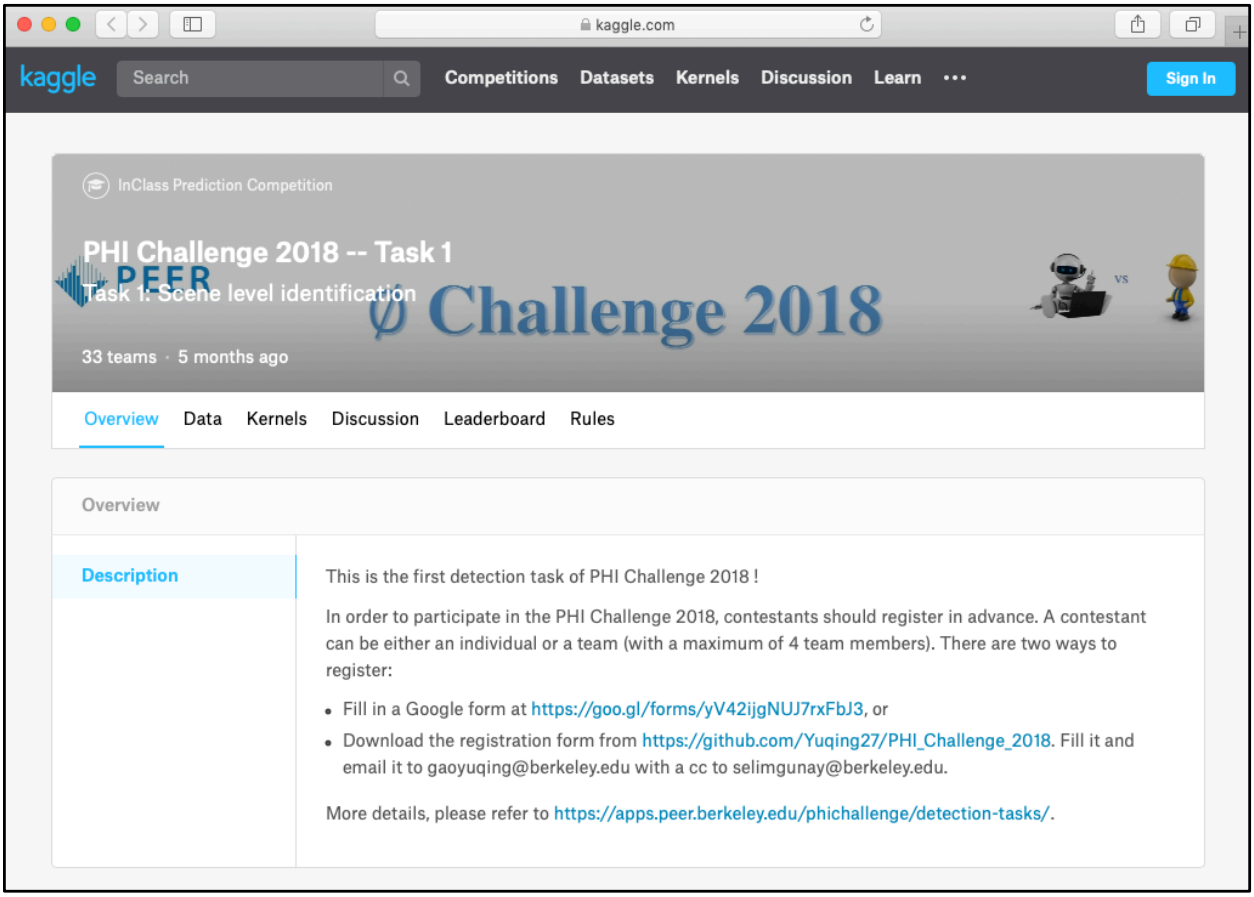


Figure 7.4: Homepage of Task 1 scene level identification of the 2018 ϕ-Net Challenge

- Task 3 (Spalling condition): `https://www.kaggle.com/c/phi2018task3`
- Task 4 (Material type): `https://www.kaggle.com/c/phi2018task4`
- Task 5 (Collapse mode): `https://www.kaggle.com/c/phi2018task5`
- Task 6 (Component type): `https://www.kaggle.com/c/phi2018task6`
- Task 7 (Damage level): `https://www.kaggle.com/c/phi2018task7`
- Task 8 (Damage type): `https://www.kaggle.com/c/phi2018task8`

To ensure good participation and submission of results, Task 1 (Scene level) and Task 2 (Damage state) were mandatory in the call for participation. Only after the participants submitted the results for Tasks 1 and 2 at least once, they were given access to the remaining tasks. These first two tasks were judged as the most basic detection tasks in vision-based SHM according to the ϕ-Net framework, refer to Fig. 4.1. After obtaining full access to all

the tasks, the participating teams could choose and perform some or all of the tasks that were of interest to them. The winners from each of the two pools of participants were determined via individual performance in each task and via the overall performance of all tasks.

Data download

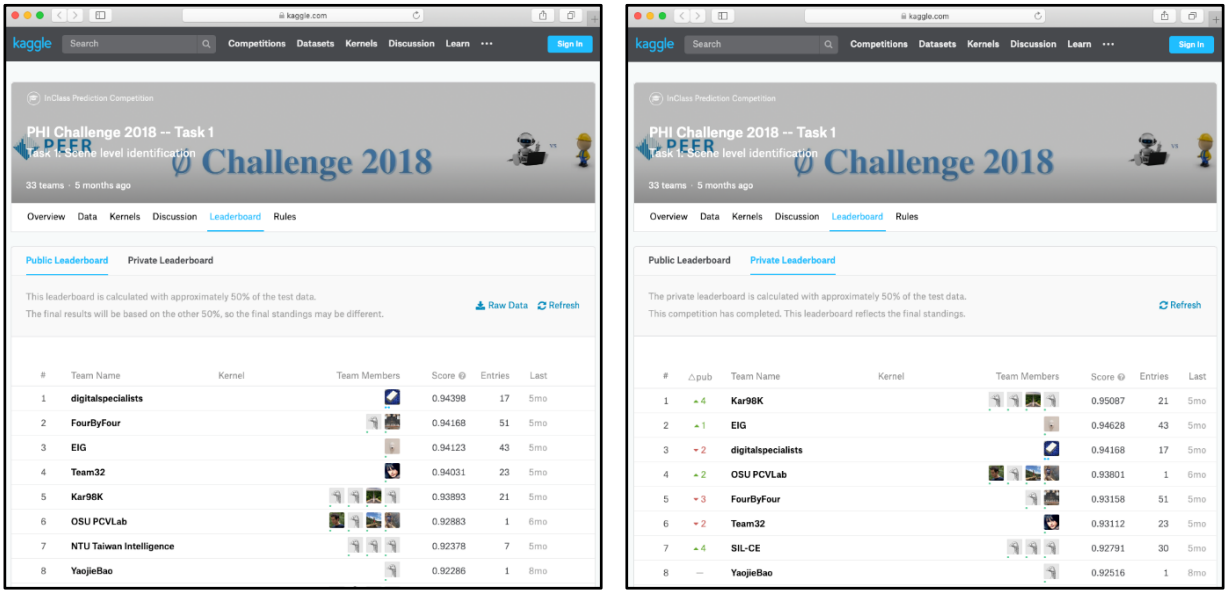
Participants were able to download the data in the **Data** tab from each task's webpage. There were four types of files in each task: **X_train.npy**, **y_train.npy**, **X_test.npy**, and **sample_submission.csv**.

- **X_train.npy** is the training data, with array shape: number of training data, 224, 224, 3.
- **y_train.npy** is the training label, whose value field is integer: 0, 1 for binary classification, 0, 1, 2 for three-class classification, and 0, 1, 2, 3 for four-class classification. Refer to the Kaggle page for the meaning of each value for each task.
- **X_test.npy** is the test data, with array shape: number of test data, 224, 224, 3.
- **sample_submission.csv** is the template for submission. The first column contains the data index, and the second column contains the prediction results. For each detection task, contestants needed to replace the second column with their own prediction results.

Results submission and leaderboard

As mentioned above, Kaggle provides an effective platform for hosting online challenges. It automatically collects the results submitted by each team and shows the ranking with scores. The teams submitted their predictions through the **Submit Prediction** tab and viewed their rankings on the **Leaderboard** page. In order to avoid excessive submissions, the submission of an individual was considered a "team" submission, and the number of submissions per team was limited to a maximum of four submissions per day.

There were two leaderboards: a public leaderboard and a private leaderboard. The former was calculated with approximately 50% of the test data, while the latter was calculated with all test data and reflected the final score. Participants only had access to the public leaderboard. Screenshots of both leaderboards for Task 1 are shown in Fig. 7.5. A comparison of the left image to the right one shows that the true scores and rankings slightly changed, e.g., No. 1 in the public board became No. 3 in the final ranking, and No. 5 in the public board became No. 1 in the final ranking. Giving participants access to only the public leaderboard increased the interest and motivated the teams to compete with one another. It is noted that the top-ranked team in the public leaderboard may not have been the "real" winner of the task. It should be stated that blindly pursuing the best performance on the public leaderboard may lead to over-fitting issues.



(a) Public leaderboard (b) Private leaderboard

Figure 7.5: Rank of the scores in the Kaggle leaderboards for the 2018 ϕ-Net Challenge

Statistics of the ϕ-Net Challenge

Sixty-eight teams applied, and thirty-four teams submitted at least one entry, for a 50% final submission rate. Considering the resource-consuming process of conducting big-data analysis and given the short time period between the announcement of the challenge and the submission deadline, this was viewed as a very encouraging level of participation. Each team was required to submit the results via the Kaggle pages (e.g., `https://www.kaggle.com/c/phi2018task1`). The statistics of the submissions by all the teams for each detection task are shown in Fig. 7.6. As mentioned above, completing Tasks 1 and 2 were mandatory for access to the remaining six tasks and received the most submissions, where almost half of the teams submitted their first several predictions. However, the optional Tasks 3 to 8 still received nearly twenty team submissions each.

The final results for the individual tasks (evaluated by the overall accuracy using Eq. 2.7) as obtained from the private leaderboard are listed in Table 7.1. It is noted that the coefficient of variation (CoV) is defined by the ratio of the standard deviation σ to the mean μ. The distributions of all results are presented in Figs 7.7a to 7.7h. It is noted that the data used in the ϕ-Net Challenge were culled from the beta version of the finalized ϕ-Net dataset. Thus, the final results of the challenge are only for reference purposes and cannot be used for calibration or considered as benchmark results.

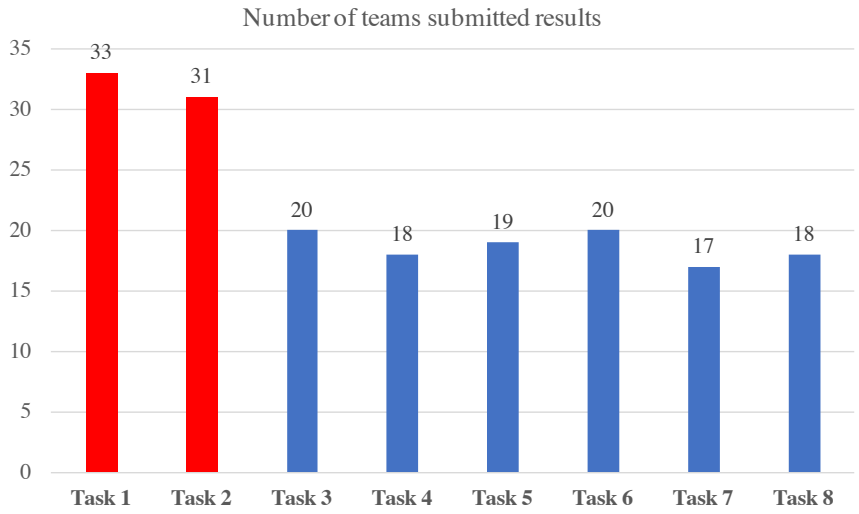


Figure 7.6: Submissions by all teams for each task of the 2018 ϕ-Net Challenge

Table 7.1: Final results (%) of all tasks of the 2018 ϕ-Net Challenge

Task id	**Task name**	**Highest**	**Lowest**	**Mean μ**	**Std σ**	**CoV (%)**
1	Scene level	95.09	31.82	89.25	5.57	6.24
2	Damage state	91.08	47.16	79.28	8.31	10.48
3	Spalling condition	82.77	71.84	79.04	2.45	3.10
4	Material type	99.82	94.84	98.16	1.34	1.37
5	Collapse mode	70.77	46.15	62.91	5.28	8.39
6	Component type	76.29	64.44	71.61	3.30	4.61
7	Damage level	79.94	55.01	66.23	5.28	7.97
8	Damage type	76.90	36.78	64.89	8.56	13.19

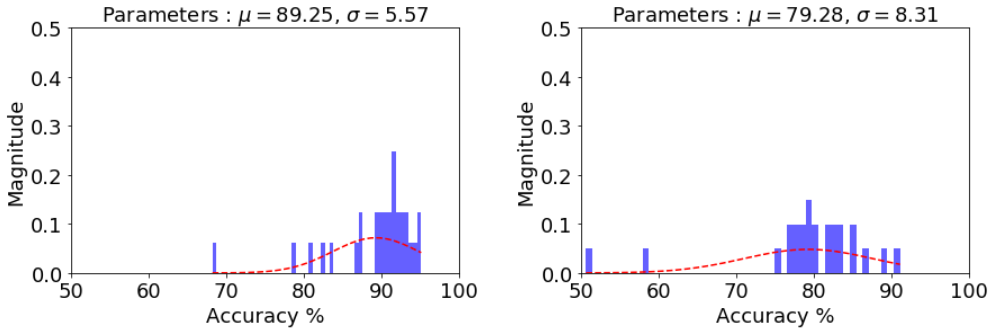
Scoring of the ϕ-Net Challenge

For the overall performance evaluation, a special metric, Eq. 7.1, was developed. It considered the natural difficulty of tasks with both prior knowledge and the performance of the teams.

$$Score = \sum_i Weight_i \cdot Test\ Accuracy_i \tag{7.1}$$

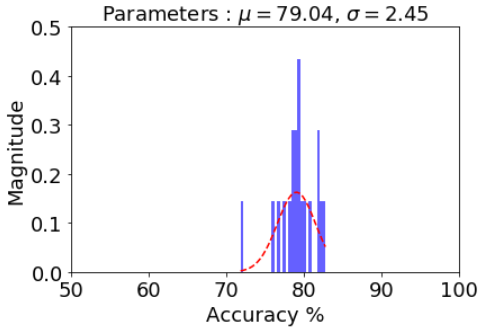
$$Weight_i = \frac{e^{(100-\mu_i)/\sum_j (100-\mu_j)} \cdot e^{\sigma_i/\sum_j \sigma_j} \cdot \phi_i}{\sum_k e^{(100-\mu_k)/\sum_j (100-\mu_j)} \cdot e^{\sigma_k/\sum_j \sigma_j} \cdot \phi_k} \tag{7.2}$$

This metric computed the weighted average of the score (accuracy) from each task where the weights, Eq. 7.2, were based on the distributions of the performance of the teams as

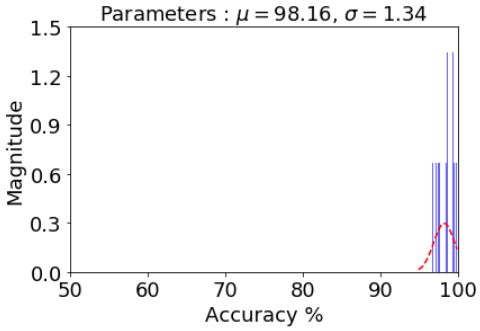


(a) Task 1: Scene level

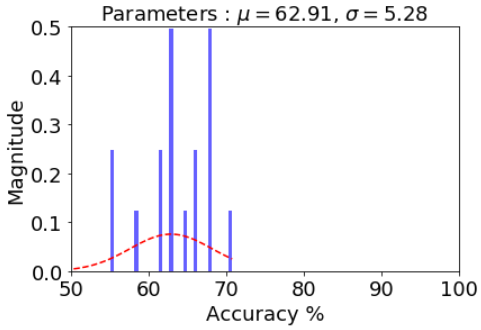
(b) Task 2: Damage state



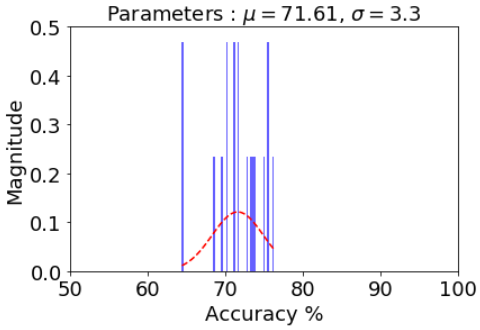
(c) Task 3: Spalling condition



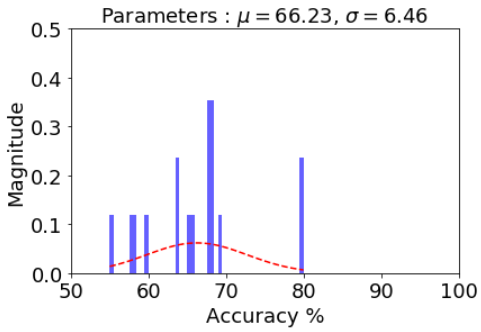
(d) Task 4: Material type



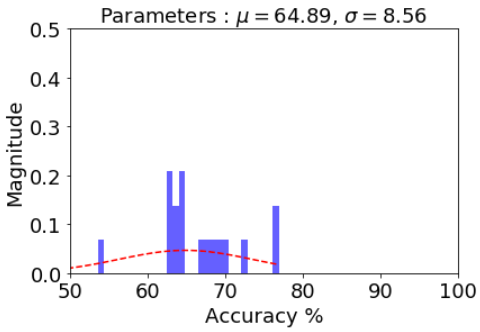
(e) Task 5: Collapse mode



(f) Task 6: Component type



(g) Task 7: Damage level



(h) Task 8: Damage type

Figure 7.7: Final results of submissions by teams for each task of the 2018 ϕ-Net Challenge

reflected by the distribution parameters (μ and σ), and the prior expected task difficulty factor ϕ. The μ and σ for each task were fitted using the final challenge results data, refer to Fig. 7.7, and ϕ values were determined by the technical committee members with domain knowledge. It is noted that in the final submissions of Tasks 1 and 2, there were several invalid results, e.g., results close to 31.82% (Task 1) and 47.16% (Task 2), which were due to one team simply predicting all test data as one category. This resulted in a close-to-random guess for a relatively balanced dataset for three-class and binary classification problems, respectively. Thus, while using the distribution parameters to compute the weights, some invalid submissions in Tasks 1 and 2 were ruled out.

Intuitively, with a higher average accuracy and a smaller standard deviation (Std), the task became easier. Thus, $100 - \mu_i$ represented the difficulty of the task (the larger the value, the harder the task), and the magnitude of σ was also positively correlated to the difficulty level. Normalizing these two terms and taking their exponential, i.e., $e^{(100-\mu_i)/\sum_j (100-\mu_j)}$ and $e^{\sigma_i/\sum_j \sigma_j}$, may result in values slightly greater than 1, considered as amplification factors. Multiplying them together and with the difficulty factor ϕ, the normalization led to the final weights for each task, which summed up to 1. Thus, in some sense the final score computed by Eq. 7.1 represented an average accuracy for all tasks. In summary, according to Eqs 7.1 & 7.2, weights for the overall performance were computed as listed in Table 7.2.

Table 7.2: Combination weights considering task difficulty and distribution parameters

Task id	**Task name**	**Mean μ (%)**	**Std σ (%)**	ϕ	**Weight**
1	Scene level	89.25	5.57	50	0.130103498
2	Damage state	79.28	8.31	30	0.088120542
3	Spalling condition	79.04	2.45	30	0.076233949
4	Material type	98.16	1.34	30	0.067004295
5	Collapse mode[1]	62.91	5.28	5	0.014852228
6	Component type	71.61	3.30	50	0.134993156
7	Damage level	66.23	5.28	80	0.233489716
8	Damage type	64.89	8.56	80	0.255202616

[1] Due to two issues in Task 5, namely (1) several duplicated and mislabeled images, i.e., two identical images with different labels, and (2) data leakage, i.e., several identical images in both training and test datasets, the difficulty factor ϕ in Task 5 was taken as a low value of 5.

Based on a comparison of the submitted predictions and the ground truth, the winners were identified by the contest judges in two categories: Computer Science/Data Science (CS/DATA) and Engineering (Non-CS/DATA). The top-3 overall performance in all tasks, top-1 performance in a single task, and best technical report were awarded for both categories. According to the weights computed above, the top-3 winners in the overall performance were

evaluated and are listed in Table 7.3. For more details about winners, refer to the challenge official website, i.e., `https://apps.peer.berkeley.edu/phichallenge/`.

Table 7.3: Top-3 overall performance of the 2018 ϕ-Net Challenge

Rank	**CS/DATA**		**Non-CS/DATA**	
	Team name	Score	Team name	Score
1st place	Kar98K	82.57	Stanford EIG	77.93
2nd place	digitalspecialists	82.56	SIL-CE	77.29
3rd place	OSU PCVLab	75.82	FourByFour-UCLA	76.97

Chapter 8

Visual Interpretation Methods of Deep Convolutional Neural Network

In Chapters 5 to 7, promising results were achieved using DL-based methods in vision-based SHM problems. However, the internal working principle in DL is hard to understand by a human, and is treated as a "black box". To conquer this problem, recently CV researchers are striving to develop visual interpretation methods, e.g., guided backpropagation (GBP) [119], class activation map (CAM) [146], gradient-weighted CAM (Grad-CAM), and Guided Grad-CAM [105], to understand the network behavior through visualizing the backward signal or the activated features. This dissertation, which is focused on the vision-based SHM, presents a comprehensive exploration towards the performance of these interpretation approaches in basic SHM vision tasks. Before diving into the experiments, several basics and background information about the DL visual interpretation methods are introduced in the following sections of this chapter.

For simplicity, some notations are redefined specifically for this chapter, which they may differ from those used in Chapters 2 & 3. In the forward pass of the CNN, F^l denotes the output from the l-th layer before the activation function, e.g., ReLU. A^l represents the activated feature maps from F^l, e.g., $A^l = ReLU(F^l)$, where only positive entries in F^l are passed to A^l. S is the class score vector computed by the deep CNN, and S^c is one entry of it for a particular class $c \in \{1, 2, ..., C\}$ where C is the total number of classes of the task at hand. In the backward pass, G^l denotes the gradient of the output with respect to A^l in the l-th layer, i.e., $G^l = \partial S / \partial A^l$. When the signal further passes back through the activation function, the restoring gradient to shallow layers is denoted as R^l, where only positive gradients can be passed back. The entry (i, j) of R^l is computed in Eq. 8.1, using the indicator function $\mathbb{1}(x)$, Eq. 8.2, refer to the example in Fig. 8.1. It is noted that the used activation function herein, ReLU, is an element-wise operation.

$$R^l_{i,j} = \mathbb{1}(G^l_{i,j} > 0) \cdot G^l_{i,j} \tag{8.1}$$

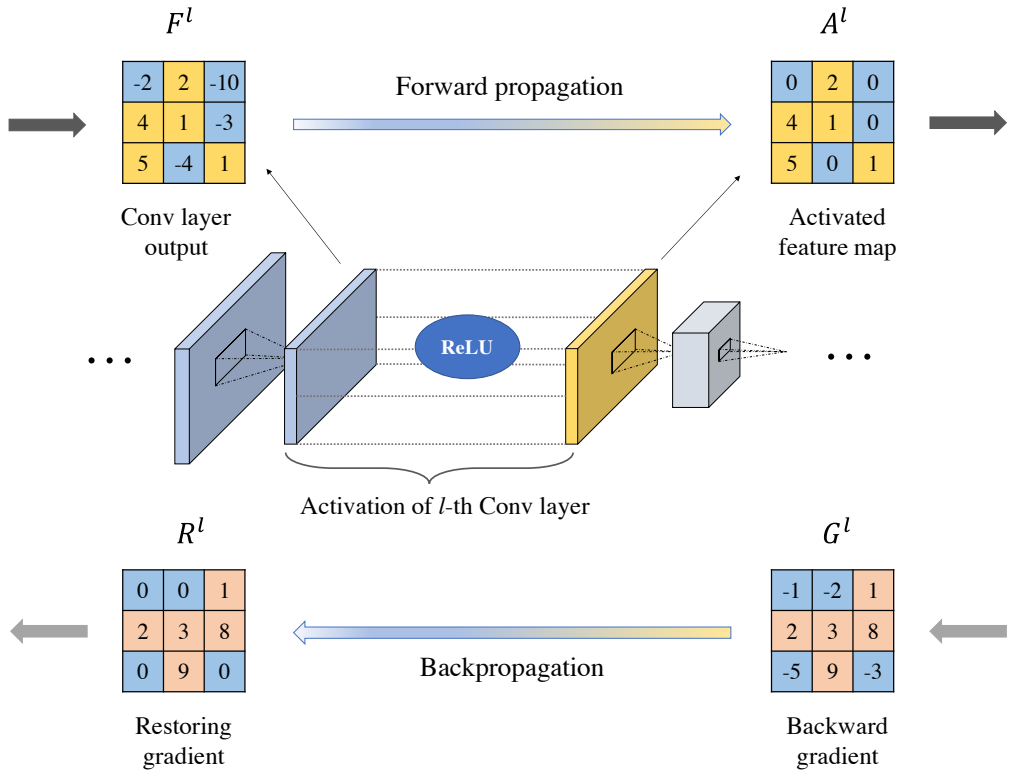


Figure 8.1: Illustration of the forward and backward signal propagation in the CNN

$$\mathbb{1}(x) = \begin{cases} 1 & \text{Event } x \text{ is } True \\ 0 & \text{Otherwise} \end{cases} \tag{8.2}$$

8.1 Guided-backpropagation (GBP)

The formulation of the GBP absorbs the characteristics of both backpropagation (BP) and deconvolutional network (Deconvnet) [140] and considers the contributions from both forward and backward passes. It is defined such that while computing the restoring gradient of the l-th layer R^l, only the entry (i, j) in the backward gradient signal with both positive gradient value $G^l_{i,j}$ and positive activation value $F^l_{i,j}$ in the forward Conv output can be passed back to the shallow layers, refer to Eq. 8.3 and Fig. 8.2.

$$R^l_{i,j} = \mathbb{1}(F^l_{i,j} > 0) \cdot \mathbb{1}(G^l_{i,j} > 0) \cdot G^l_{i,j} \tag{8.3}$$

This is reasonable because only positive activation contributes to the class score S and positive gradient contributes to the change of the parameter weights. By zeroing out unrelated information, GBP can generate clearer saliency maps as discussed in [119].

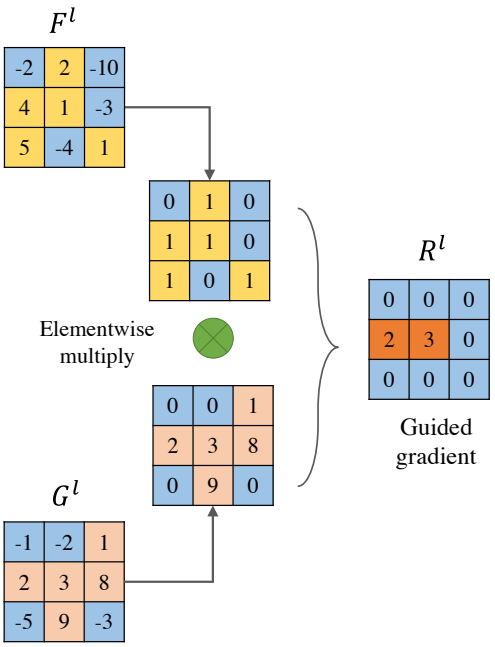


Figure 8.2: Example of the formulation of restoring gradient in GBP

8.2 Class Activation Map (CAM)

Instead of visualizing back-passing gradient like GBP, CAM presents the interpretation by a weighted sum of feature maps from one selected layer in the network, Fig. 8.3. As mentioned above, performing CAM needs to modify the original model by replacing the subsequent layers of the feature maps with global average pooling (GAP) and one fc-layer. Suppose it is intended to interpret by using feature maps of the l-th layer. Firstly, GAP projects the N activated feature maps of A^l to an N-dimensional vector, which acts as a dimension reduction. Subsequently, this vector is fully connected to the output layer which generates the class score vector S with a weight matrix W. Suppose S has C classes, i.e., the size of W is $N \times C$, and the $N \times 1$ weight vector with respect to the particular class c is denoted as W^c. Thus, the new class score S^c for one particular class c can be computed as a weighted sum of the feature vector and W^c, Eq. 8.4.

$$S^c = \sum_n W_n^c \sum_{i,j} A_{i,j;n}^l \tag{8.4}$$

where for $n \in \{1, 2, ..., N\}$, W_n^c represents the weight of the n-th feature map of A^l with respect to class c, as a scalar, $A_{i,j;n}^l$ is the entry with spatial location (i, j) in the n-th feature map of A^l. $\sum_{i,j} A_{i,j;n}^l$ is a shorthand notation of $\sum_i \sum_j A_{i,j;n}^l$ representing the GAP operation over a single feature map ignoring some constant multipliers, whose output is a scalar.

Finally, the modified model is retrained for one epoch, and then the weight matrix W is obtained. Furthermore, by summing over the product of the W_n^c and A_n^l among the N feature

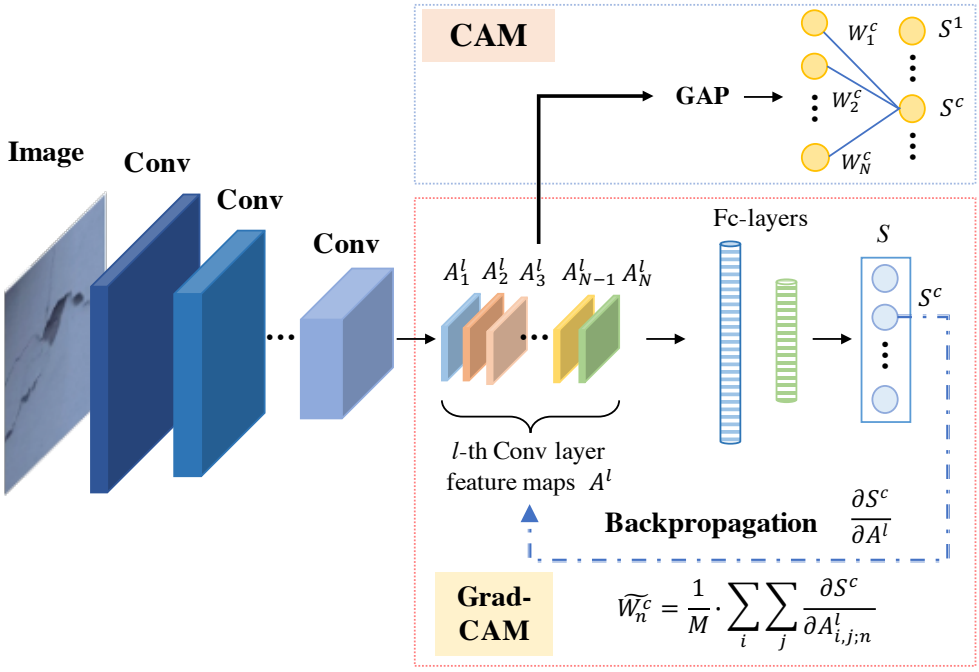


Figure 8.3: Illustration of generating saliency map of CAM and Grad-CAM

maps, the saliency map L_{CAM}^c is generated, Eq. 8.5, indicating the activation of features towards class c. Herein, the $\cdot$ operation denotes as scalar multiplication, i.e., multiplying the scalar W_n^c with each entry in the matrix A_n^l. It is noted that L_{CAM}^c in Eq. 8.5 has the same size as the feature map, and it can be further up-sampled (usually bi-linear interpolation) to a pixel-scale having the same height and width as those of the input image for a better visualization.

$$L_{CAM}^c = \sum_n W_n^c \cdot A_n^l \tag{8.5}$$

8.3 Gradient-weighted CAM (Grad-CAM)

In order to avoid the architectural modification in CAM, Grad-CAM is proposed [105]. Grad-CAM maintains the raw CNN structure and modifies the way by which weights, expressed here as $\widetilde{W_n^c}$, are calculated by using backward gradient signal. The weights are computed from GAP operation on the partial derivatives of the class score S^c over the feature maps A^l with a constant multiplier M, the total number of entries in A^l, Eq. 8.6 and Fig. 8.3.

$$\widetilde{W_n^c} = \frac{1}{M} \sum_{i,j} \frac{\partial S^c}{A_{i,j;n}^l} \tag{8.6}$$

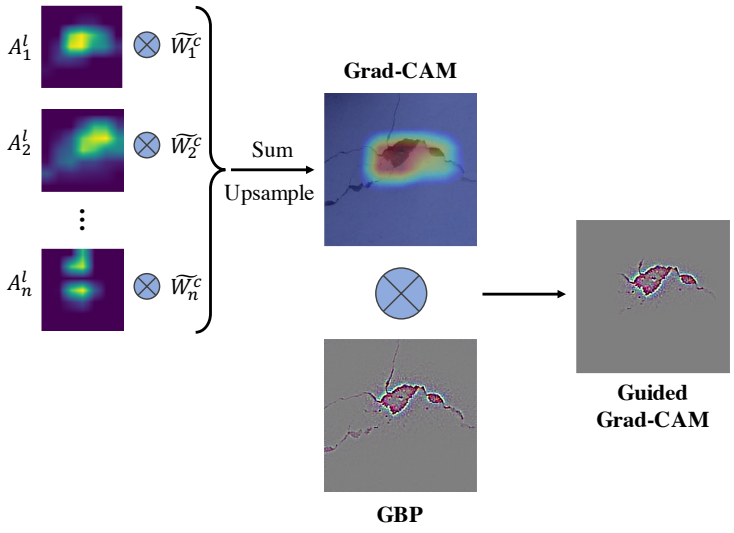


Figure 8.4: Illustration of generating saliency map of Guided Grad-CAM

Similar to Eq. 8.5, the saliency map of Grad-CAM $L^c_{Grad-CAM}$ is obtained by weighted summation over the N feature maps as follows,

$$L^c_{Grad-CAM} = \sum_n \widetilde{W}^c_n \cdot A^l_n \tag{8.7}$$

Finally, by up-sampling $L^c_{Grad-CAM}$ to the image size, a pixel-scale saliency map is generated.

8.4 Guided Grad-CAM

Further elementwise multiplication of the $L^c_{Grad-CAM}$ with the saliency map generated by GBP, one obtains a more fine-grained visualization result, which is known as the Guided Grad-CAM, Fig. 8.4. As a fusion of GBP and Grad-CAM, the saliency map in the Guided Grad-CAM is both class-discriminative and in high-resolution.

8.5 Local Interpretable Model-Agnostic Explanations (LIME)

Other than the above methods using the concepts of gradient-based or feature reconstruction, the local interpretable model-agnostic explanations (LIME) method [99] attempts to understand how the predictions change by segmenting and perturbing the input data and then to determine the important regions with positive and negative contributions towards the predictions. Moreover, LIME is model-agnostic and not specified for one type of model or problem.

In this method, more discussions are related to the image classification. Therefore, the procedure of the image-based LIME is introduced as follows:

1. Make segments by the quick shifting clustering method [126] as super-pixels[1] or instances.
2. Perturb instances as new samples by adding random noise or hiding some parts.
3. Use the perturbed new samples to perform classification based on the pre-trained CNN models and obtain weights and probabilities for each instance correspondingly.
4. Highlight instances with positive weights towards a specific class, as they give the explanations why the model chooses this class.

In the final step, usually green and red colors are used to represent the positive and negative impacts on the predictions, respectively.

[1] A super-pixel is defined as a group of pixels that share common characteristics, e.g., pixel intensity.

Chapter 9

Interpreting Convolutional Neural Network in Structural Vision Tasks

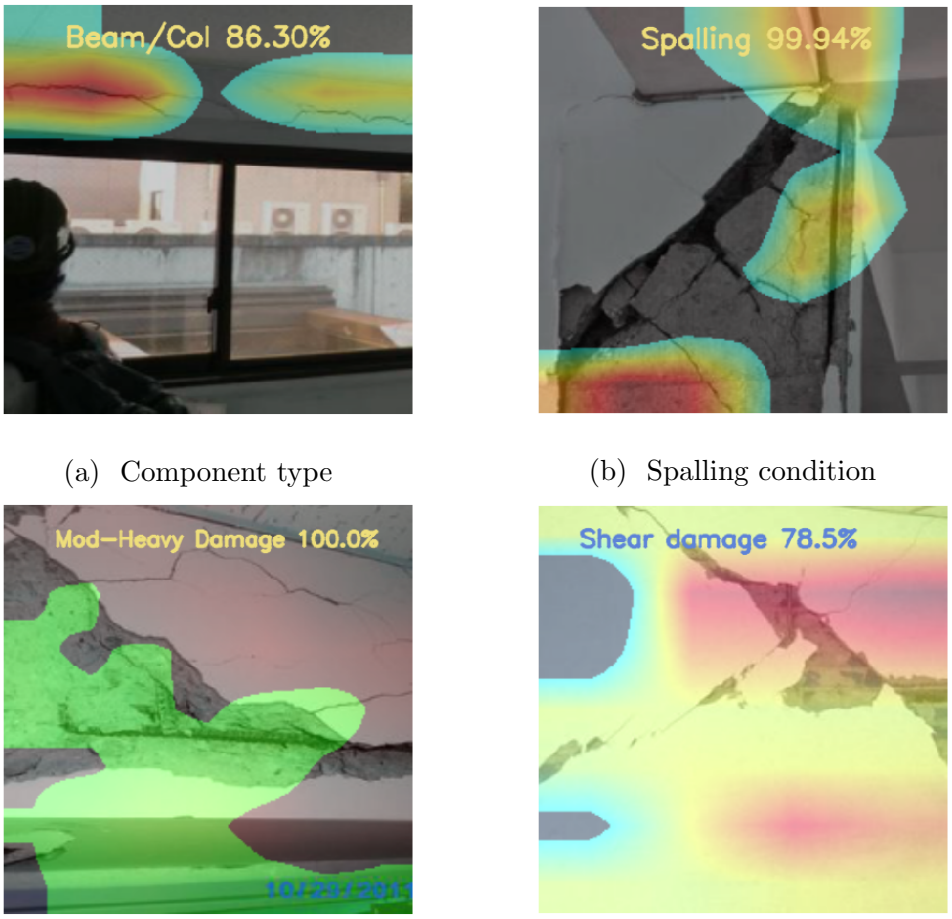
Chapter 8 reviewed several popular visual interpretation methods, and in this chapter, numerical experiments are conducted to apply these methods in interpreting how the DL model works in structural vision tasks. With some sense of understanding in the DL working principle, more self-assured DL models can be adopted in practical vision-based SHM scenarios.

9.1 Preliminary Interpretation of the Structural Vision Tasks

Before conducting a comprehensive set of experiments, several preliminary investigations were made to acquire a general idea of interpreting DL models in structural vision tasks. Therefore, two types of quick applications were considered using the well-trained models in the TL experiments, namely: (1) applying CAM [146] for the best-trained adapted VGGNet models, and (2) applying LIME [99] model for the best-trained ResNet models to evaluate the model performance based on generated saliency map, which is represented by heatmaps.

CAM for VGGNet

Fig. 9.1 shows several successful results by CAM from the best adapted VGG-type model (Table 5.9) in four tasks of the TL experiments, where the most active region in the heatmap focuses on the target. The heatmap region in Fig. 9.1a not only covers the damaged part but also presents the envelope of the beam shape. In Fig. 9.1b to 9.1d, the heatmap regions cover the damaged parts consistent with the possible damage extent defined by human experts. On the other hand, there also exist few failure cases. Fig. 9.2a indicates a wrong prediction (shear damage is the ground truth but the network only estimates 12.3% probability for the damage type to be the shear damage). Here, the CAM provides a questionable explanation



(a) Component type

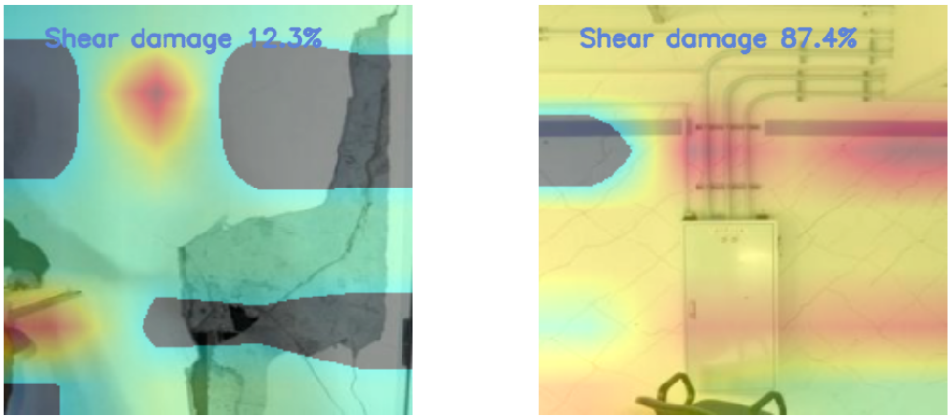
(b) Spalling condition

(c) Damage level

(d) Damage type

Figure 9.1: CAM for VGGNet in four tasks

for this damage type, where the two most active areas correspond to the wrong places, especially for the left area, which is actually a *pen* not a crack. In Fig. 9.2b, the prediction is correct, but one of the activation areas corresponds to some *pipes* instead of damage. Thus, even though the partial CAM results are useful and, as discussed in previous chapters, the overall prediction accuracy for the damage level was very high, there still exist risks to produce misleading results for other images due to these questionable patterns. Moreover, the accuracy in the damage type classification was not high enough, where the heat maps shown in Fig. 9.2 also reveal similar observations of low confidence of the model predictions of the damage type task.



(a) Incorrect prediction (b) Correct prediction

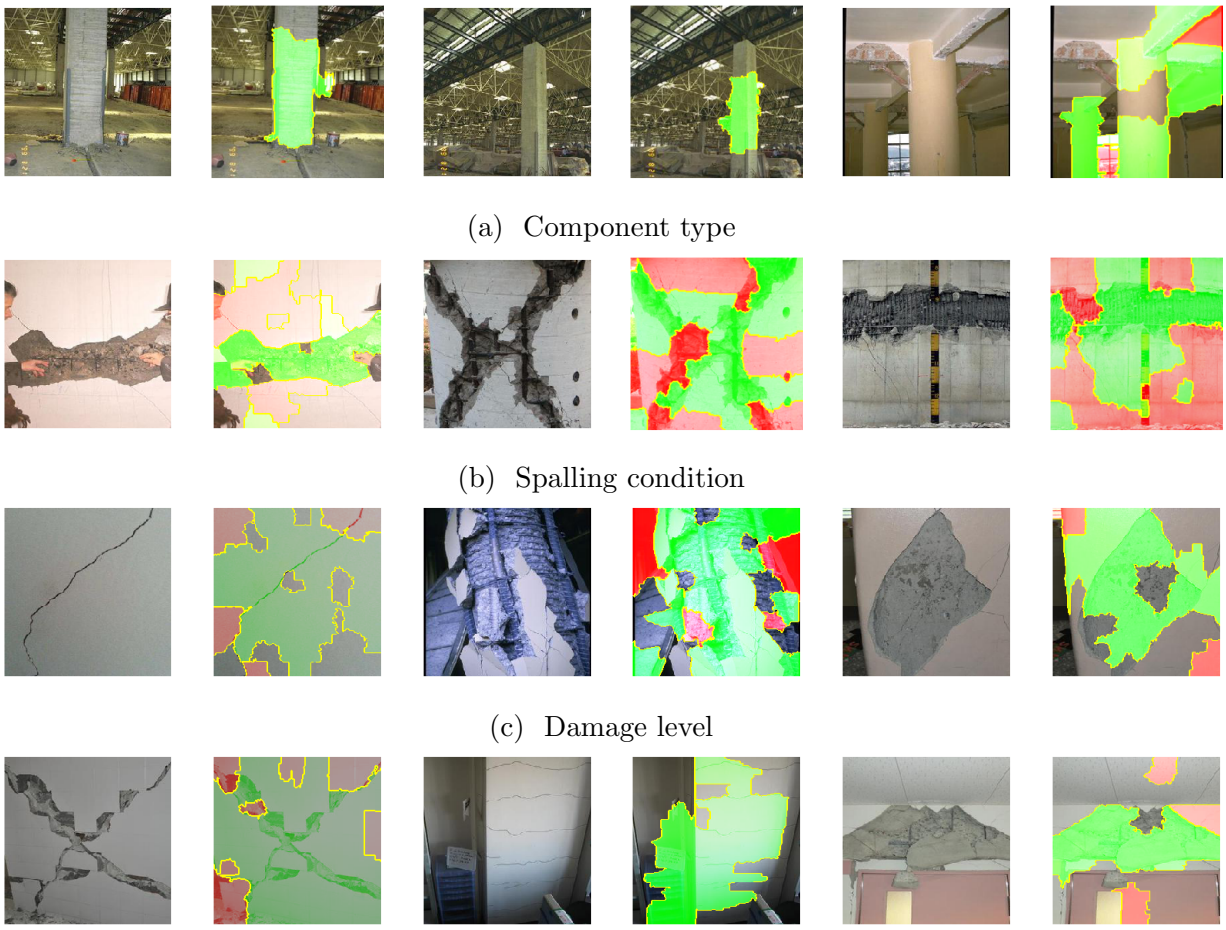
Figure 9.2: Failure cases with CAM questionable explanations

LIME in ResNet

Fig. 9.3 presents several successful explanations for the ResNet-50 model by LIME towards four tasks, where the comparisons are made between the original image (left) and the one covered by heatmap (right). It is worth noting that for the binary cases, Figs 9.3a & 9.3b, the highlighted parts also illustrate the boundaries for a column or spalling areas, which is exactly how engineers most probably will interpret the images. For the multi-class cases, Figs 9.3c & 9.3d, even though it is still unknown how the ResNet-50 makes decisions on the damage level and type, the highlighted parts cover most of the damaged parts, indicating that the network still focuses on the right target which makes the model more reasonable. For simplicity, only results of the best-trained ResNet-50 models (Table 5.9) were plotted here, and similar performance observations were obtained in the fine-tuned ResNet-101 but with slightly refined and more accurate boundaries. Therefore, combing the results in Table 5.9 with the LIME interpretations, the ResNet models not only can achieve satisfactory prediction accuracy but also have well explainable properties with informative LIME visualizations.

9.2 Experimental Objectives

Section 9.1 presented two cases of directly using two types of interpretation methods in explaining the output of the trained DL models in structural vision tasks. However, the active areas in the saliency maps (heatmaps) are somewhat ambiguous, and only limited portions of the prediction results can be reasonably presented and explained. This situation raises two questions towards the coarse and less universal interpretation results: (1) is it due to defects in the model recognition ability which is not well-trained and not intelligent enough? or (2) is it due to the limitation of the applied visual interpretation methods? To



(a) Component type

(b) Spalling condition

(c) Damage level

(d) Damage type

Figure 9.3: LIME for ResNet-50 in four tasks

address the first question, using the best pre-trained models from the benchmark experiments is a feasible solution. This is because from the evaluation metric of the overall accuracy, the benchmark models achieved promising results especially in easy tasks (Tasks 1 to 4) based on both relatively large-scale training and test datasets. Accordingly, the results with respect to the model performance are reliable. To address the the second question, Grad-CAM and Guided Grad-CAM [105] are thought of as the improved and more general versions of CAM which can instead be reliably applied into the planned virtual experiments.

Based on these two solutions, more comprehensive investigations via experiments should be pursued. Therefore, five numerical experiments with respect to several key factors influencing the deep CNN models were conducted as follow:

1. To complement the above validation experiments, Grad-CAM and Guided Grad-CAM were implemented to validate the performance of the well-trained adapted VGGNet

models from the ϕ-Net benchmark experiments in one basic (relatively easy) vision task, namely spalling condition (Task 3) detection. The saliency maps of both Grad-CAM and Guided Grad-CAM for the predicted "spalling" class label were produced and analyzed.

2. During the model training procedure, under an ideal situation, the deep CNN gradually converges to local minima with decreasing loss through gradient updating. To better understand the learning process of the model used in (1), the observations in the saliency maps produced by GBP, Grad-CAM, and Guided Grad-CAM at several key epochs during training were visualized and discussed.

3. Zeiler & Fergus [140] and Selvaraju et al. [105] explored the visualization of the CNN layer features at different depths, which indicated that the deep layers capture high-level semantic information and have better interpretable results than the shallow ones. Similarly, in the TL experiments in Chapter 5, it is found that the deep layers are more task-dependent and contain essential information for damage recognition. Thus, the visual explanation of the activated feature maps for "spalling" extracted at different network depths, i.e., from shallow to deep, was investigated to understand the findings in these TL experiments.

4. During training, several strategies may help deep CNN achieve a better performance with limited data, e.g., TL and DA. Results from the previously discussed TL and benchmark experiments of the ϕ-Net demonstrated that TL helps the model achieve remarkable performance enhancement in the spalling condition task. On the contrary, some results from the VGGNet models indicate the ineffectiveness of the DA where the accuracy does not change if solely using DA. Moreover, using TL and DA simultaneously rendered worse results than using TL only. Thus, comparisons of the saliency maps between the baseline model trained from scratch (BL), BL using TL (TL), and BL using both TL and DA (TL-DA) were conducted and discussed.

5. The above investigations are within the scope of concrete spalling detection, which is a basic and non-expert task where identifying the occurrence of spalling does not require much professional knowledge. As an extension, a similar exploration was conducted on a more complex task with higher semantic abstraction, namely damage type task (Task 8) identification. It involves domain knowledge, e.g., the definition of flexural and shear damages with specified visual patterns originating from the respective ductile and brittle failure modes of the structural component in a seismic design [82]. Those patterns are not so visually obvious and sometimes even experts may find it challenging to make accurate decisions. In other words, these patterns present the foundation for the label semantic meanings in a more abstract manner.

The presented validation experiments herein were aimed to address the following questions:

1. Can Grad-CAM and Guided Grad-CAM methods work in explaining the results of the pre-trained models for basic vision SHM tasks?
2. How does the model learn during training?
3. What is the contribution from different depths of the model?
4. What are the influences of different training techniques, e.g., TL and DA?
5. Can deep CNN learn well during more abstract tasks?

It is noted that the major contribution of this part of the dissertation is not to develop new visual interpretation methods, but to introduce existing ones for better understanding of the deep CNN trained model for image-based SHM problems and to investigate several key factors for establishing more convincing DL models. This is an attempt towards increasing the level of confidence of engineers in DL models, further improving the model performance, and finally confidently applying these DL models to practical problems. Based on the conducted literature review, there is no similar comprehensive study in the data-driven SHM approach until now.

9.3 Experimental Setup

According to the above objectives, five types of experiments were conducted, namely (1) a general view of the interpretation methods on the basic spalling condition task, (2) investigation on key epochs during training, (3) exploration of features at different network depths, (4) influence of TL and DA, and (5) exploration in interpreting abstract (expert) tasks to classify the damage types. Datasets of both spalling conditions and damage types were subsets of the benchmark ϕ-Net dataset (Chapter 4), and the used deep CNN models were the best adapted VGG-16 networks trained in the benchmark experiments for these two tasks, Table 6.9.

In the interpretation analysis, there exist four possible outcomes, namely *correct prediction with reasonable visualization* (CorRes), *correct prediction with unreasonable visualization* (CorUnr), *incorrect prediction with reasonable visualization* (IncRes), and *incorrect prediction with unreasonable visualization* (IncUnr). For example, IncRes means the prediction is wrong but the focus and active region(s) of the saliency map still point to the damage location(s). In the following experiments, most of the shown results are CorRes and IncRes. However, CorUnr and IncUnr results are also presented for model diagnosis in some cases. For the purpose of investigating the model's generalization performance, the saliency maps are mainly generated based on the test images.

Since Grad-CAM is class-discriminative, a single informative class can be selected to understand what is the evidence that the used DL model is effective for decision making. However, unlike previous Grad-CAM studies using natural objects or vision patterns with certain rules to follow, the label "non-spalling" (NSP) and "undamaged state" (UD) in SHM are more abstract, and to recognize them in a classification task is more like an occurrence

detection problem. Thus, saliency maps with respect to these labels may not be meaningful. Take an example of NSP images, because there are no spalling patterns, the active regions in the saliency map can point to anywhere in the image, which is indeterminate and does not provide useful information for interpretation. On the contrary, if Grad-CAM generates saliency maps with respect to the label "spalling" (SP), the active regions should only point to the corresponding spalled areas, which is determinate. As a result, images with NSP and UD labels are not considered in this study. Besides the general settings, the detailed configurations for each experiment are presented in the following sections.

Interpretation on a basic task

In this experiment, the best VGG-16 network adopting TL pre-trained from the ϕ-Net benchmark Task 3 experiments (Table 5.9) was used herein, and denoted as VGG-TL for short. The saliency maps were generated from the feature maps of the last Conv layer. The evaluation metric used for the pre-trained model is the overall accuracy (Eq. 2.7). The values of training and test accuracy of the model were 99.2% and 81.6%, respectively. To further analyze the bias in the test set, the confusion matrix is presented in Fig. 9.4a.

Influence of the learning procedure

The deep CNN usually needs multiple training epochs, and the accuracy increases along with the model parameter updating. In this experiment, a few featured epochs were selected and investigated when the test accuracy increased in the training process of the VGG-TL. According to the training history in the benchmark experiment (Fig. 6.3), selected epochs were 1, 2, 3, 5, 8, 11, and 14, whose test accuracy values were 68.5%, 78.6%, 79.1%, 80.0%, 80.6%, 81.5%, and 81.6%, respectively. The model achieved the best test accuracy at epoch 14. In addition, if denoting correct prediction for SP as true positive (TP), correct prediction for NSP as true negative (TN), incorrect prediction for SP as false positive (FP), and incorrect prediction for NSP as false negative (FN), the corresponding *true positive rate* (TPR), Eq. 2.10, and *true negative rate* (TNR), Eq. 9.1, are reported in Table 9.1 together with the *overall test accuracy* (Eq. 2.8) to better check the model bias.

$$TNR = \frac{TN}{TN + FP} \tag{9.1}$$

Influence of the network depth

According to the architecture description in Table 5.2, the VGG-type network was made from five Conv blocks followed by multiple fc-layers. In this experiment, saliency maps of the VGG-TL were generated based on the last layer (max pooling) of each Conv block, from shallow to deep, which represents the learning performance of the network at different depths.

Table 9.1: Overall test accuracy, TPR & TNR (%) of the best adpated VGG-16 at different training epochs

Epoch	**0**	**1**	**2**	**3**	**5**	**8**	**11**	**14**
Accuracy	63.0[1]	68.5	78.6	79.1	80.0	80.6	81.5	81.6
TPR	0.0	87.7	84.8	81.9	88.4	74.2	78.4	85.5
TNR	100.0	58.1	74.6	77.2	75.1	84.6	83.3	79.3

[1] Because the dataset is not strictly balanced (there are more NSP images than SP ones), at epoch 0, the model simply classifies all images as NSP, and the overall accuracy is higher than 50%, which would have occurred in a balanced dataset.

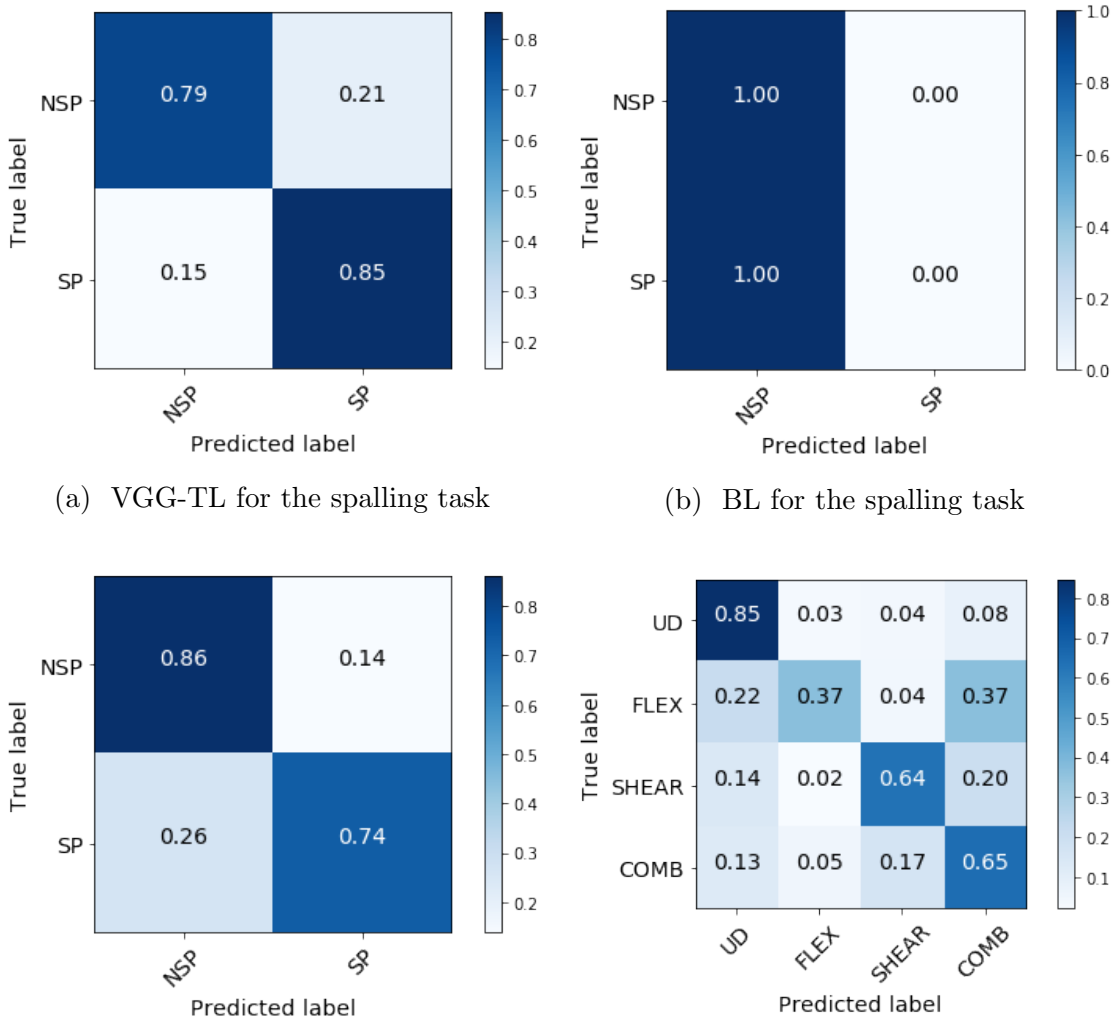
Influence of the training techniques

Since the above experiments already explored the saliency maps of the model using TL, to investigate the influence of the TL, visual results of the BL were produced for comparison. The confusion matrix of the BL is shown in Fig. 9.4b, where all data were predicted as NSP, indicating a severely biased model. Moreover, according to Figs 6.3a & 6.3b, the training loss and accuracy of the BL were observed to be unchangeable during the whole training procedure.

Besides, DA is another common strategy used in training. However, in the benchmark experiment of Task 3, while using DA in training BL, the model was still untrainable with the same poor accuracy, Figs 6.3a & 6.3b. Thus, in this experiment, only VGG-TL-DA was used and compared to VGG-TL to discuss the influence of the DA. The test accuracy of VGG-TL-DA was 81.5%, and the confusion matrix is shown in Fig. 9.4c, which shows that the model gained performance improvement in identifying NSP images with the trade-off in accuracy decrease in detecting SP.

Influence of the semantic abstraction

In this experiment, similar work was repeated to investigate the interpretation on the damage type task via VGG-TL. According to the benchmark experiments, the values of training and test accuracy of VGG-TL were 98.5% and 70.7%, respectively, and the confusion matrix is shown in Fig. 9.4d. It is noted that in Fig. 9.4d, the test accuracy for label “flexural damage” (FLEX) was very low and that for labels “shear damage” (SHEAR) and “combined damage” (COMB) were also not high enough for practical usage. Thus, the model itself is unreliable. However, besides having some sense on the results interpretation, the saliency maps provide information for model diagnosis. To complement the analysis, few saliency maps of examples in the training dataset are also presented.



(a) VGG-TL for the spalling task

(b) BL for the spalling task

(c) VGG-TL-DA for the spalling task

(d) VGG-TL for the damage type task

Figure 9.4: The confusion matrices for the adapted VGG-16 under different scenarios

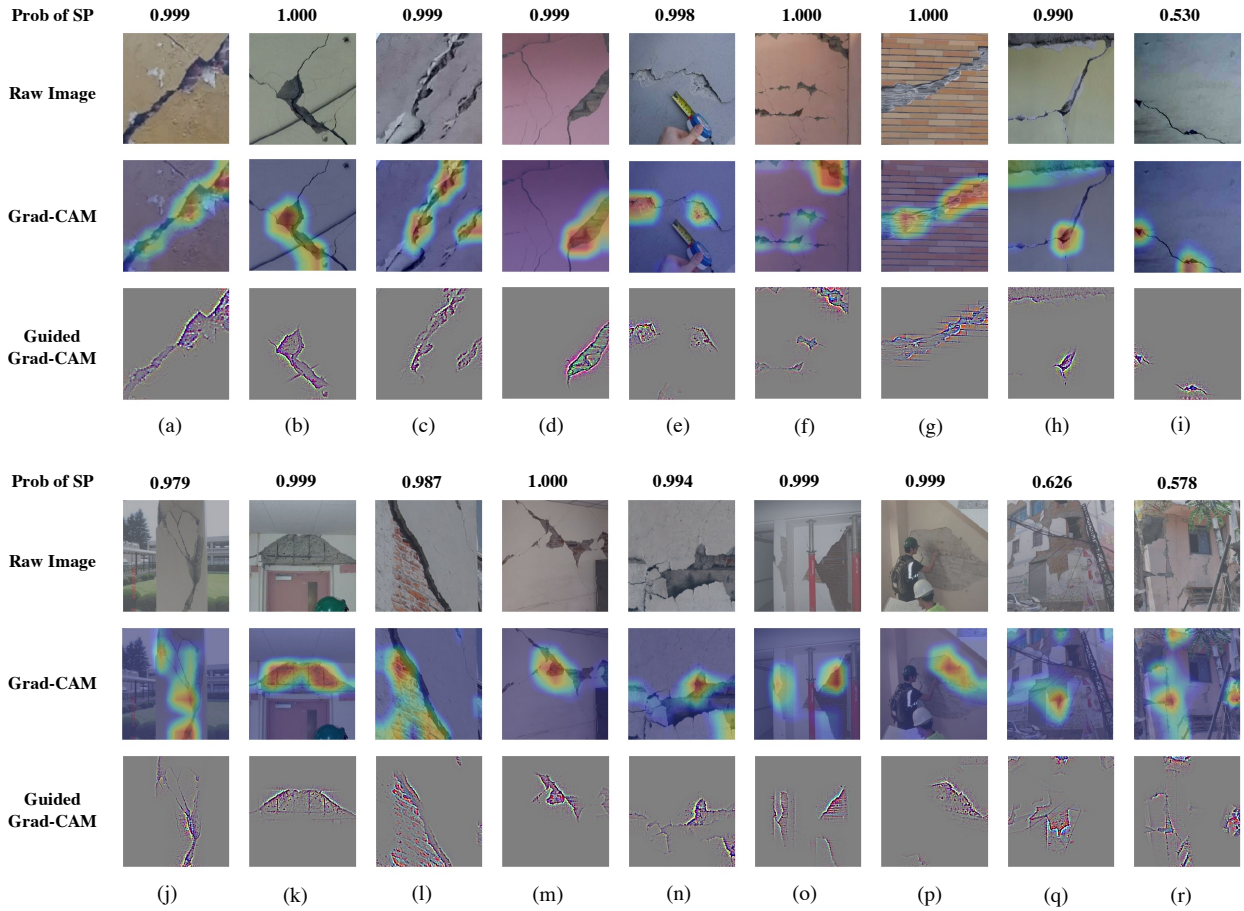


Figure 9.5: Saliency maps of Grad-CAM and Guided Grad-CAM in predicting the SP class with correct predictions (CorRes)

9.4 Experimental Results & Discussions

Interpretation on a basic task

According to the experimental settings, the saliency maps of Grad-CAM and Guided Grad-CAM with respect to label SP are presented in Fig. 9.5, all of which are CorRes. Examples of IncRes and IncUnr are shown in Fig. 9.6. Through comprehensive observations, there were very limited CorUnr and few IncUnr results among the thousands of the training and test data. The saliency maps furnished good visual explanations in most cases in this basic (non-expert) task, where the active regions for class label were very close to human judgement. Therefore, this experiment indicates the robustness of the Grad-CAM for this type of basic vision task in the image-based SHM. Moreover, several findings and conclusions of this experiment are discussed in the following paragraphs.

The first important observation is that the interpretable quality of Grad-CAM and Guided

Grad-CAM is more related to the scale of the image description (also known as the scene level), and model recognition ability and preference. From the label definition in Section 4.1, the spalling condition task is usually conducted on the pixel or the object level images, representing images taken from close or mid-range distances towards the structural component. It is easy to observe that most close-up images have better visualization results, Figs 9.5a to 9.5l, where the heatmaps almost cover the entire spalled area and also accurately capture multiple areas. There only exist very few slightly worse cases, e.g., Figs 9.5m & 9.5n, which are at the pixel level but only highlight partial spalled areas. While selecting images with a boarder view, i.e., object level images, the heatmaps still can locate the correct damage areas. However, more results with only highlighting partial damage areas are produced, Figs 9.5o to 9.5r. From previous studies, this observation can be partially addressed by the limitation of Grad-CAM, where its heatmaps often do not cover the entire object in completeness [17]. In addition to this reason, it is speculated herein that another reason is attributed to the recognition ability and prediction preference of the trained VGG-type model more than the Grad-CAM itself. Using TL in the VGG-type model usually requires to resize the images to the fixed 224×224 resolutions as inputs, and this results in the same size of the respective field[1] of each Conv layer for images corresponding to both pixel and object levels. Under the same size of the respective field, it is easier to recognize the SP class with easier visual patterns on the pixel level, which describes these patterns with less noise and more distinctive features compared to those on the object level. Therefore, larger areas in the feature maps are more likely to be activated in a close-up image. However, incomplete active regions do not negatively affect the rationality of this interpretation method in any significant manner. Because it is still reasonable to predict the SP class by only finding partial spalled area instead of the entire region, which is similar to the usual human behavior in decision making. Based on this reasoning, the active areas can be reasonably studied for better understanding of the recognition ability of the VGG-type model in the following experiments.

The second observation is that the trained model is prone to predict the image as SP once identifying the ellipse-like spots within the entire spalled area, especially in more evident cases of Figs 9.5a to 9.5g. For images with irregular spalling shapes, the Grad-CAM saliency maps still indicate ellipse-like active areas even with limited coverage. For example, in Fig. 9.5k, the actual spalled shape is close to a trapezoid, and the model recognizes it as SP with two elliptical regions near the two hypotenuses of the trapezoid. This observation indicates that the model is more sensitive to the ellipse-like visual patterns which are common characteristics among many SP training images. It is expected that this trained model can achieve good and explainable recognition on other unseen images with regular ellipse-like spalling patterns.

Thirdly, the trained DL model is also capable of identifying tiny spalled area, but it loses confidence (lower probability) in decision-making. For example, in Fig. 9.5i, the probability of SP decreases to 0.53, and it increases the risk of missed detection. Thus, this observation further supports the conclusion in the first observation about the difficulty of the current trained model to identify spalling among different scene levels.

[1]The receptive field in CNN refers to the part of the image/feature map visible to one filter at a time.

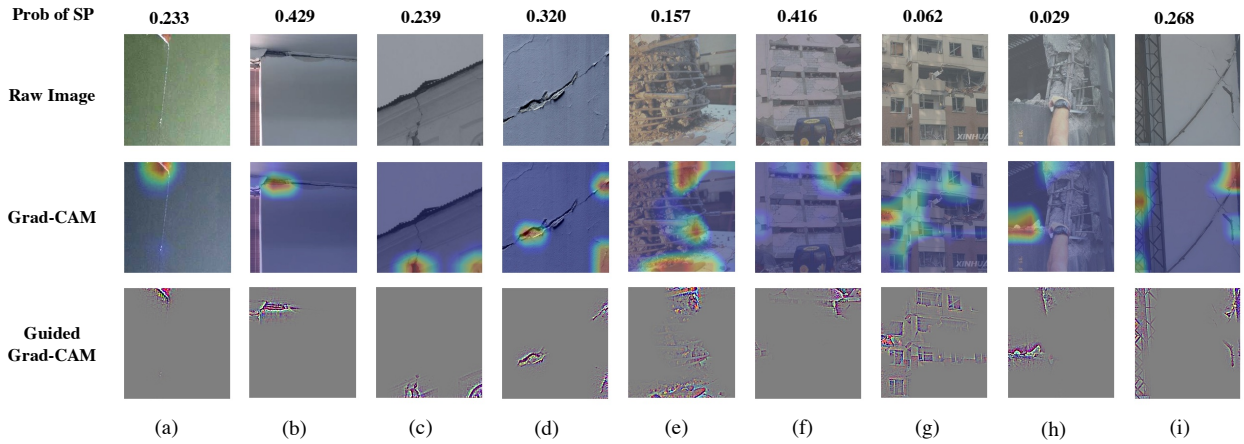


Figure 9.6: Saliency maps of Grad-CAM and Guided Grad-CAM in predicting SP with incorrect predictions (IncRes & IncUnr)

Compared to coarse heatmaps generated by Grad-CAM, the saliency maps by Guided Grad-CAM present a high-resolution and pixel-scale visualization, which clearly highlight the pixels that positively activate the prediction results. Beyond the precise damage location, the results in the Guided Grad-CAM also illustrate the boundary (edge) of the spalled area along with cracks. Moreover, the textures of the spalling are highlighted in some cases, e.g., exposed concreted in Fig. 9.5k and masonry in Fig. 9.5l, which provide the inference bases for the VGG-type models.

Besides discussions from CorRes, the results of IncRes, Figs 9.6a to 9.6e, and IncUnr, Figs 9.6f to 9.6i, are also studied for model diagnoses. Figs 9.6a to 9.6d present similar tiny spalling patterns to Fig. 9.5i. However, herein the model made wrong predictions as NSP. This is expected to take place analogous to the observation in Fig. 9.5i, where the probability was low and close to 0.5, which represents a random guess probability. Therefore, such model limitations result into increasing the FN predictions. Fig. 9.6e describes a heavily spalled column, where the visual pattern is much different from most images and there is no clear boundary between the intact and the spalled parts. From the saliency maps of Figs 9.6a to 9.6e, the VGG-TL does capture some key features, e.g., texture and location of the spalled area. However, the model made the wrong predictions. This indicates that the trained model is capable of finding the spalling-related features, but it encounters the problem of finding the correct relationship between these features and the label due to only learning from very few of these types of images, which results into a lack of generalization in the prediction. IncUnr results in Figs 9.6f & 9.6g are mainly due to these images having a much boarder scope, namely in the object level or even close to the structural level. This broad scope brings sharp differences between most SP images in the training and test sets. Thus, the model is incapable of effectively generalizing to the test images. Similarly, Fig. 9.6h is also a rare heavily spalled case and Fig. 9.6i contains some jamming objects, e.g., wire mesh and electric cables, which make the model incapable of correctly focusing on the target and leading to

incorrect predictions.

From the above analysis, a better understanding of the trained VGG-TL model for the prediction of the SP class is achieved. Several conclusions with respect to the model diagnosis are listed as follows:

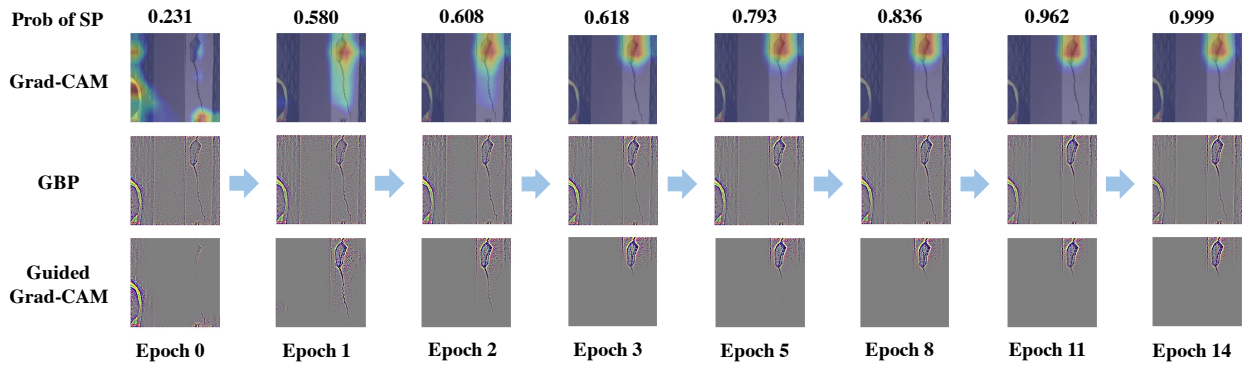
- The current trained VGG-type model has both accurate and explainable recognition ability on the pixel level images, but its performance degrades with the increasing distance from the object, which indicates that the model cannot generalize well to all scene levels.
- The model is still less accurate in identifying tiny and uncommon spalling patterns and for images containing more noise.

The above two issues can be alleviated by retraining the model with more images. Besides, another easier way is to separate the data with more fine-grained scene levels, and then learn the spalling patterns separately based on different scene levels.

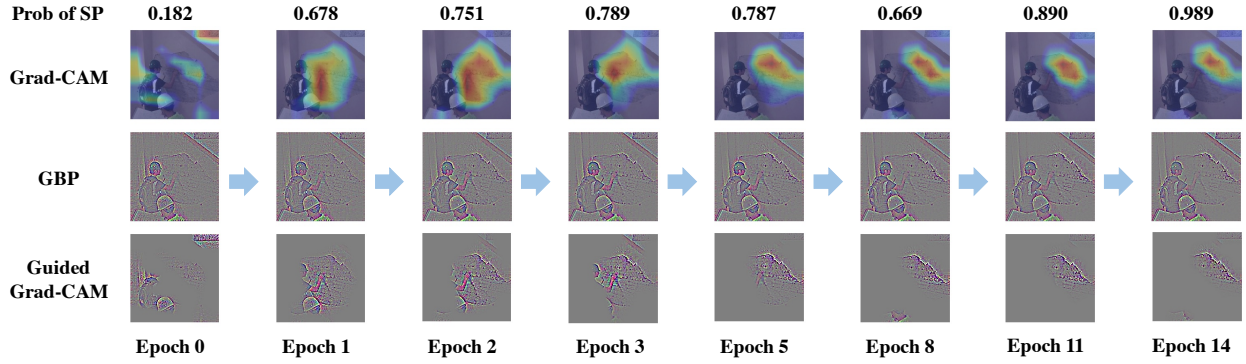
Influence of the learning procedure

In this experiment, the saliency maps of the VGG-TL model for different training epochs were studied and two examples are presented. Fig. 9.7a is CorRes and has a good heatmap coverage that is converging towards epoch 14 when achieving the best test accuracy. The shown rows of the saliency maps illustrate the learning procedure of the model. Before fine-tuning the model (epoch 0), only using pre-trained weights from ImageNet [24] to capture image features, the model is totally biased and predicts all data as NSP. The active regions of Grad-CAM and Guided Grad-CAM with respect to SP are deviated from the damage areas. On the contrary, as class-indiscriminative, the saliency maps of GBP highlight all important pixels the model capture, where the edges of the spalled area and cracks are well-captured. However, these GBP maps also highlight other edges, e.g., the boundary of the column and wire meshes. This observation indicates that the ImageNet pre-trained model is good at detecting line-like edges, which is the most dominating characteristic of cracks and spalling areas. With transferring knowledge to the target domain of the SP images, the model starts to discriminate spalling-related edges and ignore the irrelevant ones. From epochs 1 to 14, it is observed that the active regions of the heatmap are approaching the correct spalled locations with accurate coverage, and the probability of predicting SP is increasing, which indicates the gaining of discriminative abilities.

Besides, the model is learning rapidly based on TL and it obtains a significant performance enhancement in detecting SP (TPR 87.7%) only with one epoch training. Moreover, the model is getting smarter and linking the prediction with the visual patterns of cracking and spalling, where both cracks and spalled areas are highlighted in the Grad-CAM results. As the training process continues, the model learns a better relationship between the label SP and the visual pattern with an increasing confidence (higher probability of SP), where it waives



(a) An example with good heatmap coverage



(b) An example with incomplete heatmap coverage

Figure 9.7: Two examples of CorRes results at different training epochs. Note: Epoch 0 means directly conducting prediction based on the ImageNet pre-trained weights without fine-tuning, and the VGG-TL model achieves the best test accuracy at epoch 14.

the occurrence of cracks and only focuses on the spalled areas. Unlike Grad-CAM and Guided Grad-CAM, the GBP results are less variant due to class-indiscriminative property. Thus, the model is always maintaining attentions on these edges of the features and is gradually learning to only use the most class-discriminative part to help predict the label. This property is also similar to what human experts do, where they take a broad view of the image first, find some salient features, and finally select the most class-discriminative ones for decision making.

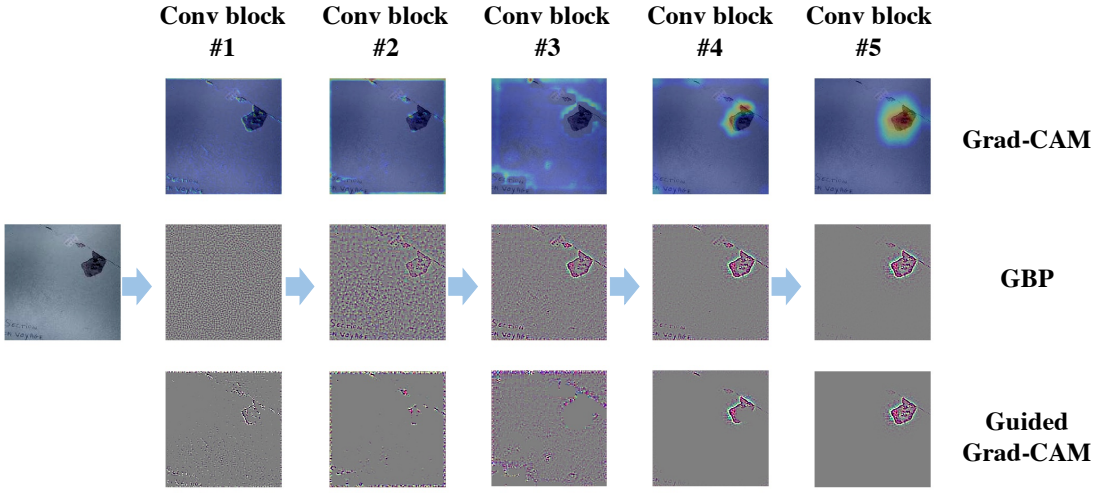
Fig. 9.7b illustrates the learning procedure of a CorRes example with incomplete heatmap coverage. Besides similar conclusions to those from Fig. 9.7a, this example further demonstrates the inference that the incompleteness of the saliency maps is mainly due to the recognition ability and the preference of the model. The active regions of both Grad-CAM and Guided Grad-CAM are large at epochs 1, 2, 3, and 5, but an obvious trend of shrinkage towards the top right corner is observed, and it continues until achieving the best accuracy at

epoch 14. From the saliency maps of the Guided Grad-CAM at epochs 1 to 5, the edges of irrelevant objects, like the shown human arm, also contribute to the prediction, and the model tends to waive these influences by shrinking the discriminative regions. As concluded in the first experiment about interpreting a basic task, the VGG-TL model is prone to be active towards an ellipse-like shape, which is convex and easy to learn. On the contrary, in Fig. 9.7b, the complete spalled area without the human arm forms a concave shape, which contradicts the above-mentioned convex shape. Even though such concave shape can be represented by activating multiple elliptical regions, it is usually harder to achieve a complete coverage compared to the convex shapes. Because the model is learned by a gradient descent method, which is a type of a *greedy approach*, where gradients are along the steepest descent direction where the loss function decreases the most, refer to Eq. 3.8. In this manner, compared to bypassing the pixel region of the human arm to form multiple ellipse-like active regions, it is easier and faster to only move and shrink the active regions towards the upper corner of the spalled area with one ellipse-like region. To support these interpretations, the transit from epochs 3 to 5 indicates an evidence that the model mostly waives the influence of irrelevant objects, i.e., the pixel region of the human arm disappears in the Guided Grad-CAM and accordingly the TPR of the model increases from 81.9% to 88.4%. Even though the model undergoes a moderate drop in confidence (probability of SP decreases from 0.787 to 0.669) and TPR (decreases from 88.4% to 74.2%), as the learning continues, the model further learns the trend, where the covered pixels in the regions eventually lead to the most discriminative prediction with a higher confidence and relatively higher TPR, refer to epochs 11 and 12 in 9.7b. Therefore, this example presented well how to understand the learning procedure of the model, and also partially supported the previous inference about the reasons behind the incompleteness in the visual coverage of the active regions.

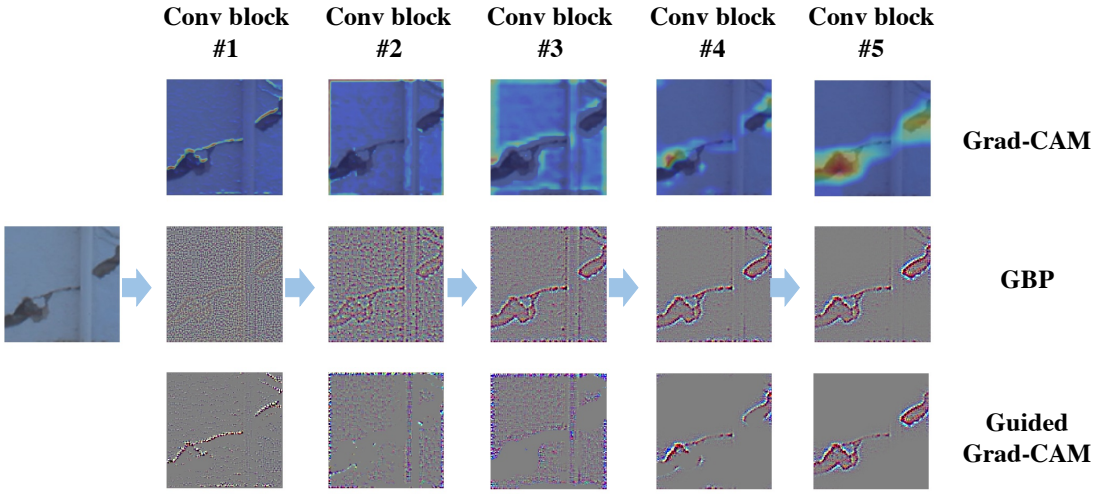
Influence of the network depth

In this experiment, the behavior of the deep CNN is explored along its depth. The saliency maps of two CorRes examples from five different depths are presented in Fig. 9.8. Starting from GBP, as class-indiscriminative, it plots all important pixels contributing to predict both SP and NSP. In the shallow layers, i.e., Conv blocks #1 and #2, the saliency maps are not so informative where most pixels are highlighted and look like noise. However, starting from block #2, the saliency maps begin to show certain edge features, e.g., shape and boundary of cracks and spalled areas. When increasing the depth to block #3, the edge features of the cracking and spalling become apparent with decreasing noisy pixels. Further increasing the depth to Conv blocks #4 and #5, the edge features are more evident. This observation shows that, even in the shallow layers, the trained model is still capable of learning and identifying cracks and spalling edges. This indicates the feasibility of only using a relatively shallow CNN to identify the concrete cracking and spalling from images with less noise.

From the results of Grad-CAM and Guided Grad-CAM, shallow CNN layers, e.g., Conv blocks #1 and #2, are not sensitive in discriminating SP, where the active regions are small with limited number of activated pixels. With the increasing depth towards Conv blocks #4



(a) Example 1



(b) Example 2

Figure 9.8: Saliency maps of two CorRes examples from different network depths. Probability of SP for both (a) & (b) are 0.999.

and #5, the active regions start to focus on the correct damage locations. This observation validates the characteristics of the deep CNN that deep layers are more task-dependent, and explain well the experimental results in the TL experiments where retraining the last two Conv blocks of the VGG-type model leads to achieving significant accuracy enhancements.

Influence of the training techniques

From above three sections, visual interpretation analysis for the adapted VGG-16 model using TL was conducted and indicated the good property of TL. Now, back to the baseline model BL where the model parameters were trained from a random space or certain distributions, e.g., Gaussian. The confusion matrix of BL in Fig. 9.4b indicates a biased model, where it predicts everything as NSP, and the probability of SP for each image is as low as 0.430. Thus, from these metrics, it is concluded that the BL is problematic. The saliency maps of BL are shown in Fig. 9.9, and all results are IncUnr. The saliency maps of GBP in this case show a view of dispersion points highlighted in purple. Regardless of the discriminative property, herein, the model no longer captures any basic visual features for the damage, e.g., edge or texture, and simply randomly treats each pixel as important, which indicates a scenario of no learning. Moreover, when investigating the discriminative ability, the important regions of Grad-CAM and Guided Grad-CAM for all images are similar but focus on the margin and corner areas. Thus, they do not show any clear relationships towards either cracking or spalling. Therefore, by visualization, it is easier for users to understand and diagnose the BL model since (1) it cannot capture the damage features, and (2) it lacks the discriminative ability towards the SP label. Combining the flat training history curves (Figs 6.3a & 6.3b[2]) and the aimless saliency maps, these poor performances have one possible reason, which is attributed to a poor parameter initialization. On the contrary, while adopting TL, the model parameters were pre-trained from ImageNet instead of from a random space. The abundant information during the pre-training empowers the model with some sense of edge and texture of the natural objects, which has been validated from the results of epoch 0 in Fig. 9.7. On the basis of such basic feature capturing, the model starts to gain accurate discriminative ability along with parameter fine-tuning during the training procedure.

In the training procedure of several classification tasks, the overall accuracy (Eq. 2.7) and cross-entropy (Eq. 2.5) are commonly used as an evaluation metric and a loss function, respectively. In the comparisons between VGG-TL and VGG-TL-DA, the only 0.1% difference in the overall accuracy is deceptive, which hides the performance of TPR and TNR. From the confusion matrices in Fig. 9.4c, even though VGG-TL-DA model gains ability to detect NSP (higher TNR), its performance deteriorates in detecting SP (lower TPR), which brings more risks to practical applications. The decrease of the overall accuracy in SP increases the FP, i.e., wrong predictions for SP. However, it only leads to the increase of IncRes, which results from a comprehensive saliency maps check, especially in GBP. Thus, this preliminary check

[2]Both BL and BL-DA of the adapted VGG-16 have the same untrainable performance, so their learning curves are overlapped.

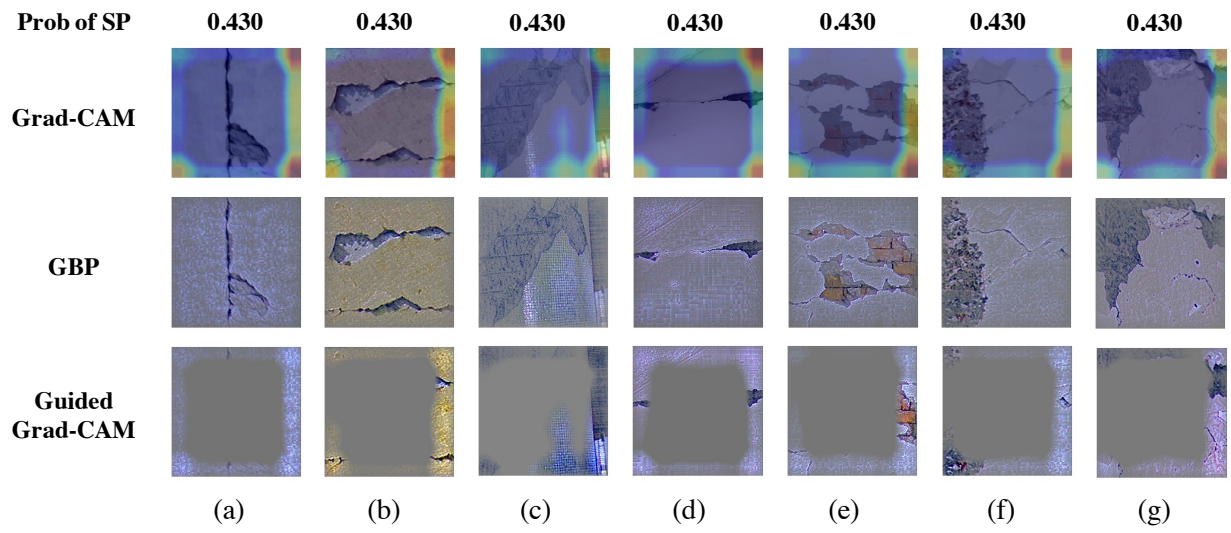


Figure 9.9: Saliency maps of BL (IncUnr)

indicates that VGG-TL-DA still captures the explainable visual damage features but loses some sense of discriminative ability in correctly detecting the SP label. In Fig. 9.10, four representative examples are plotted. Fig. 9.10a is an easy example, and the performance of VGG-TL-DA is similar to that of VGG-TL. Fig. 9.10b represents an object level SP image, which is thought to be harder than a pixel level image. It is observed that the active regions in Grad-CAM not only have the spalled area but also contains the upper wide crack, where both parts contribute to the final decision. With the use of DA, the probability of SP decreases significantly from 0.999 for VGG-TL to 0.698 for VGG-TL-DA. It is an obvious evidence that VGG-TL-DA is less discriminative for SP. Figs 9.10c & 9.10d are two other relatively difficult examples presented in Fig. 9.5. For Fig. 9.10c, Guided Grad-CAM and Grad-CAM of VGG-TL-DA produce similar visualization to those of VGG-TL, which indicates the similar incompleteness visualization in some mid/far distance images. However, the probability of SP with DA drops significantly from 0.988 for VGG-TL to 0.649 for VGG-TL-DA and it has a great risk to be a wrong prediction. This provides another evidence of the worse discriminative ability. Fig. 9.10d represents an example where the VGG-TL-DA fails in prediction and shows unreasonable heat map but VGG-TL performs perfectly on both fronts. This case further provides the evidence that VGG-TL-DA model loses the generalization in less common images and makes wrong predictions. It is noted that the example of Figs 9.10a & 9.10b are the most common cases in VGG-TL-DA. This is followed by the example of Fig. 9.10c, and finally the example of Fig. 9.10d is the least common case. The worse performance of VGG-TL-DA in the saliency maps visualization interprets well the 11% TPR downstream performance compared to VGG-TL, where the TPR of VGG-TL and the VGG-TL-DA are 85% and 74%, respectively, refer to Figs 9.4a & 9.4c).

Even though BL, VGG-TL, and VGG-TL-DA have the same source from the adapted

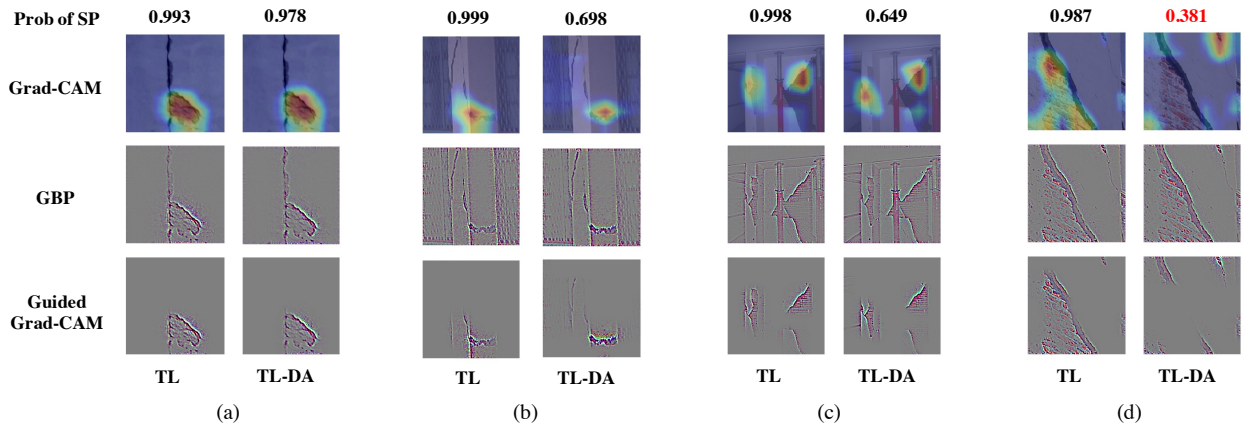


Figure 9.10: Four representative examples of saliency maps for VGG-TL and VGG-TL-DA

VGG-16 network, due to different training techniques and different initial conditions, they become totally different models. Moreover, the numerical values of the overall accuracy can be deceptive. Instead, a better understanding and comprehensive model evaluation can be obtained via a complementary saliency maps analysis.

Influence of the semantic abstraction

In this experiment, visual interpretation methods were adopted to a more complex and abstract task, which has more fine-grained labels, namely undamaged (UD), flexural (FLEX), shear (SHEAR), and combined (COMB) damage types. The saliency maps in predicting images with FLEX labels are plotted in Fig. 9.11. Due to only 37% accuracy in detecting FLEX images, wrong predictions are obtained. Unlike spalling detection, in this case, most results of Grad-CAM and Guided Grad-CAM are IncUnr and point to irrelevant pixel patches, which indicate that the VGG-TL model did not learn well the FLEX label. Examples of Figs 9.11a to 9.11c illustrate that the model is capable of capturing thin and horizontal cracks, which represent the most successful prediction cases of FLEX. However, there still exist some failure cases like Fig. 9.11e. Figs 9.11d & 9.11f are examples of damage concentrating at the ends of the component to form plastic hinges. Fig. 9.11d roughly captures the boundary of the spalled region, but Fig. 9.11f is totally wrong. In addition, the salience maps of images in training dataset are also presented. Fig. 9.11g indicates that the model did learn well in recognizing horizontal cracks during training providing the foundation for further generalizing to unseen test data. On the contrary, Figs 9.11h & 9.11i show the unreasonable active regions towards prediction even though the model makes the correct prediction with a high probability. Thus, the model did not fully understand the semantic meaning of FLEX even during training, and of course did not perform well for the test data, especially for the case of damage focusing on the component's ends. From the perspective of GBP, the edges of the cracks and spalled areas are sharp and can be somewhat easy to highlight. Thus, it is

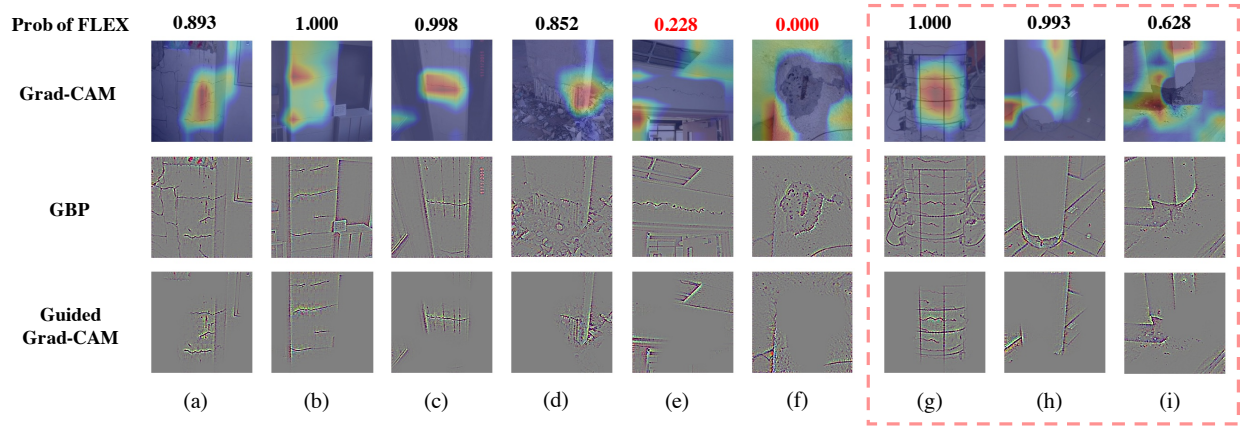


Figure 9.11: Saliency maps for FLEX. (a) to (f) from test dataset with (a) to (d) CorRes and (e) & (f) IncUnr. (g) to (i) are from training dataset with (g) CorRes and (h) & (i) CorUnr.

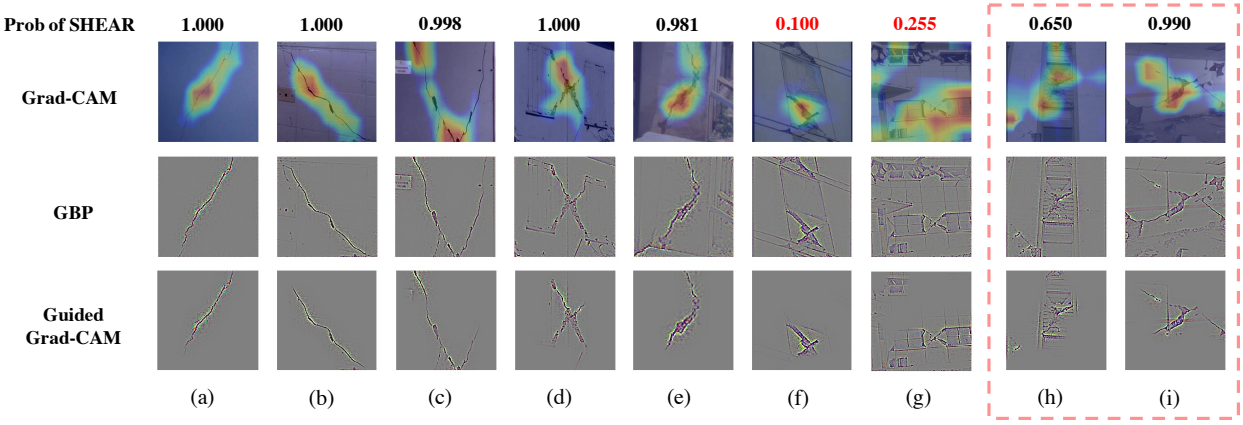


Figure 9.12: Saliency maps for SHEAR. (a) to (g) from test dataset with (a) to (e) CorRes and (f) & (g) IncRes. (h) & (i) are CorRes from training dataset.

evident that the model is still somewhat able to predict the basic damage features.

The saliency maps in predicting images with the SHEAR label are plotted in Fig. 9.12. Compared to FLEX, the SHEAR class accuracy is 64%. As a result, more CorRes (Figs 9.12a to 9.12e) and very few IncUnr results are obtained, where wrong predictions are usually corresponding to IncRes, Figs 9.12f & 9.12g. Consistent with the definition of SHEAR, the VGG-TL model is capable of detecting thin diagonal cracks in multiple directions (Figs 9.12a, 9.12b & 9.12e), "V" shape cracks (Fig. 9.12c) and "X" shape cracks (Fig. 9.12d) with a high detection accuracy and excellent saliency map visualization. From the confusion matrix in Fig. 9.4d, 20% wrong predictions of the images with the SHEAR label are misclassified as COMB. The most common characteristics of these images are belonging to the object level or have complex visual patterns, e.g., wide cracks or large spalled area as failure cases in Figs 9.12f & 9.12g. Further investigation of the training data, Figs 9.12h & 9.12i, indicates that

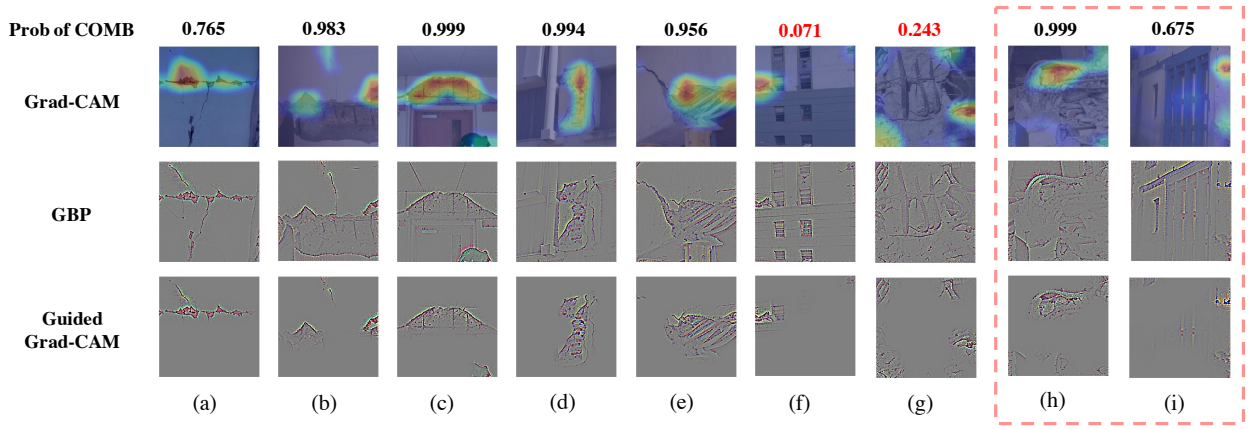


Figure 9.13: Saliency maps for COMB. (a) to (g) are from test dataset with (a) to (e) CorRes and (f) & (g) IncRes & IncUnr, respectively. (h) & (i) are CorRes from training dataset.

during training the model not only can recognize and locate damage in complex examples but also it can correctly relate the SHEAR label to diagonal cracks instead of other heavily spalled areas. This is consistent with both technical definition and human expert knowledge. Therefore, it is concluded that, the model is capable of recognizing key features of SHEAR under several typical vision patterns, especially for the pixel level images.

The saliency maps in predicting images with the COMB label are plotted in Fig. 9.13. Figs 9.13a & 9.13b illustrate that the model made the decisions based on the simultaneous occurrence of horizontal and inclined cracks with spalling patterns in some cases. This is consistent with one scenario of combined damage definition. Figs 9.13c to 9.13e show that the active areas of the saliency map by both Grad-CAM and Guided Grad-CAM focus on the heavily spalled areas, where the damage is mainly caused by complex mixture effects, which is also consistent with the COMB label definition. The IncRes case (Fig. 9.13f) represents the pool discriminative result of the model towards damage in distant images. Even in the training procedure, the confidence in the discrimination is still not high enough, Fig. 9.13i. Unlike the SHEAR label cases, there still exist certain number of IncUnr results like Fig. 9.13g, where the raw image describes a heavily damaged structural component with significantly spalled concrete and buckled reinforcement. In this case, due to irregular visual patterns and sharp differences between these patterns, the difficulty in accurately extracting the features increases. From Fig. 9.13h, the model is actually capable of predicting the correct result with an incomplete heatmap during training, where the key discriminative visual pattern is the occurrence of exposed buckled reinforcing bars. Therefore, such issue is expected to be alleviated if more of such types of images can be collected for training.

In summary, results in this experiment further emphasize the earlier statement that the numerical performance of the model is somewhat deceptive where the approximate 65% test accuracy for SHEAR and COMB does not mean the model can correctly recognize 65% of other unseen SHEAR and COMB images. Through visual interpretation analysis, it is found

that the current model learned well some specific patterns of each damage type, e.g., diagonal, "X" shape, and "V" shape cracks for SHEAR, combination of horizontal and inclined cracks and heavily spalled areas for COMB. However, it is difficult to generalize to more complex and abstract cases, e.g., when damage concentrates at the element's ends. Moreover, the results shown by Grad-CAM and Guided Grad-CAM clearly expose the model's weakness in discriminating the abstract semantic label consisting of multiple visual patterns. However, the saliency maps from the GBP indicate that the model is capable of finding basic features related to damage in most images regardless of the classes. Therefore, it is concluded that deep CNN has great potential to recognize abstract and semantic tasks, but at this stage, due to limited training data, the model lacks discriminative ability to find a clear relationship between reasonable visual features and semantic labels.

9.5 Summary

In this chapter, applications of the recent widely-used visual interpretation methods, namely GBP, Grad-CAM, and Guided Grad-CAM, in the basic tasks of vision-based SHM were investigated. Beyond the general view over the basic (non-expert) spalling condition task, several key factors affecting the performance of deep CNN were explored, namely learning procedure, learned features from different network depths, training techniques by using TL and DA, and level of semantic abstraction represented by the damage level task. Through comprehensive experiments, several key observations and conclusions are obtained and summarized as follows:

- Grad-CAM and Guided Grad-CAM performed well in interpreting well-trained deep CNN models for the spalling condition task, where most of the saliency maps produced good visual explanations. The visual interpretable qualities are related to the scene level of the images, the model recognition ability, and the model preference, where the best visual interpretable results were obtained in the pixel level SP images.

- While identifying SP, ellipse-like regions to be visual covered are usually used as evidence of model prediction. The model is also capable of finding some tiny spalled areas. However, its performance degrades with the increase of image complexity and some other small or uncommon damage patterns.

- Benefiting from TL, the VGG-TL model learned rapidly in the first few epochs. The active regions in the heatmap of Grad-CAM and Guided Grad-CAM changed to accommodate the increasing discriminative ability of the model, which may sometimes end up with an incomplete coverage of the damage area. Regardless of the class-discriminative property, the results of the GBP indicate that the model is good at capturing line-like edge features, whose property is inherited from the pre-trained ImageNet model, and these features are suitable for making judgement on the damage state or the spalling condition from a human expert perspective.

- Through investigating the model features at different depths, the dominating influence of deep layers in spalling identification is clearly illustrated. From the saliency maps of the GBP, the model started to have some sense of the damage features from shallow and intermediate layers, and the discriminative ability was achieved in the deep layers, e.g., layers in Conv block #5.
- The baseline model BL initialized from a random parameter space was shown to be untrainable. The obtained saliency maps consisted of a dispersion of points. These observations implied that the BL model is incapable of damage features extraction and also lacks the discriminative ability, which can be significantly alleviated by a better initialization methods such as using TL.
- The saliency maps emphasized the differences between the VGG-TL and the VGG-TL-DA models, where they made predictions according to different basis. The visual interpretable results are more consistent with the TPR performance instead of the deceptive overall accuracy.
- When adopted to the more abstract damage level task, worse visual results were obtained than those in the non-expert spalling task. It is concluded that with the limited training data, the current model is still not able to recognize well the abstract semantic meaning of the damage type, especially for the FLEX label. However, some saliency maps of Grad-CAM and Guide Grad-CAM show that the model still has some sense of recognizing specific damage patterns for the SHEAR and COMB damage types.
- From the perspective of the model performance, the visual explanations provided by saliency maps give more intuitions to improve the model, e.g., enlarge the training dataset, further split data into more fine-grained scene levels, and conduct classification within each level. It should be noted that besides cracks, other line-like edges were also highlighted in the saliency maps of the current model, e.g., outlines of the component and grid lines in the test specimen. Therefore, not only introducing more data but also more variety of data including noise are expected to help in improving the model robustness.

Based on the above visual interpretation analysis, three drawbacks and future studies to remedy these drawbacks are summarized as follows:

- Except for the main reason dictated by the model recognition ability (low recognition accuracy) and preference (prone to making prediction only by part of the target), the issue of incomplete active region coverage may still be related to the limitation of Grad-CAM method. It is expected to try the modified version of Grad-CAM, e.g., Grad-CAM++ [17], for a better and complete visualization.
- For brevity, only gradient-based and feature reconstruction approaches (GBP, Grad-CAM, and Guided Grad-CAM) were mainly investigated in this dissertation. Other interpretation methods, e.g., LIME [99], need more evaluations and discussions.

- Only visual interpretation and model diagnosis were discussed in this dissertation. There exist great potentials and room for better model selection metrics based on the understanding obtained by these methods. These mainly consider the physical meaning of the model, and make the predictions more transparent and accordingly convincing.

In summary, Grad-CAM and Guided Grad-CAM can provide relatively reasonable visual explanations of the basic SHM vision-based tasks. Through analysis, it is easier for human experts to understand the trained deep CNN model with these visualization methods' recognition capabilities, preferences, and limitations. Even though it is still not comprehensive, a good understanding of the deep CNN model partially relieves the black-box concerns and may also give more confidence to the human experts for consideration of CNN in practical applications. Moreover, the conclusions & extensions summarized above shed light on the great potentials of this research direction in vision-based SHM, which needs more explorations in the future.

Chapter 10

Generative Adversarial Network (GAN) for Image Generation

Chapters 4 to 9 covered the basic applications of DL and deep CNN in vision-based SHM. In this and the following two chapters, namely Chapters 10 to 12, the state-of-the-art DL technology of the Generative Adversarial Network (GAN) is investigated for the purpose of synthetic image generation.

The concept of GAN was first introduced by Goodfellow et al. [39] in 2014, which is a generative model that generates new data from learned data distribution. Unlike conventional DL models, GAN consists of two networks, namely the generator and the discriminator, where the generator creates synthetic data and the discriminator classifies an input sample as "real" or as "synthetic". GAN uses adversarial training, where each network aims to minimize the gain of the opposite side while maximizing its own. Ideally, both the generator and the discriminator converge to the *Nash equilibrium* [37], where the discriminator gives equal predictive probabilities to the real and the synthesized samples. In other words, it indicates that the quality of the synthetic images produced by the generator is as good as that of the real images. Thus, the discriminator can no long distinguish whether the data are real or synthetic.

Even though GAN is a generic approach, more successful results have been reported in computer vision tasks, e.g., synthetic (fake) image generation [12, 97] and image-to-image translation [48]. Following the discussion that started from Chapter 4, in this part of the dissertation, the scope of the introduced GAN is limited to CV applications, namely using GAN in vision-based SHM tasks. Therefore, two aspects, namely synthetic structural image generation and augmentation for classification purposes, are mainly investigated. Before diving directly to the computer experiments, the mathematical foundation, network configurations, and evaluation metrics of GAN are introduced in this chapter first.

10.1 Principles & Mathematical Formulation of GAN

As mentioned above, GAN consists of parametrized discriminator and generator, which can be any designed functions or neural networks. The training of GAN is in some sense adversarial by playing the game between the generator and the discriminator where they look forward to iteratively maximize their own outcomes and minimize the opponent's turn by turn. Fig. 10.1 illustrates the procedure of how GAN works. The generator aims to produce synthetic data to fool the discriminator. Conversely, the discriminator aims to distinguish between real data and synthetic data produced by the generator. Once the generator fools the discriminator successfully, i.e., G1 fools D1, the discriminator updates itself until it can recognize the synthetic data correctly, i.e, D1 updates itself to D2. Similarly, once the synthetic data are identified by the upgraded discriminator, the generator updates itself until it can fool the discriminator again, i.e., G1 updates to G2. Finally, the game ends when the discriminator can no longer update itself to identify the synthetic data, i.e., G3 vs D3. Under such a situation, the probability of classification of the synthetic versus the real data of the discriminator becomes about 0.5, i.e., a random guess, and the synthetic data are thought to be sufficiently close to the real data.

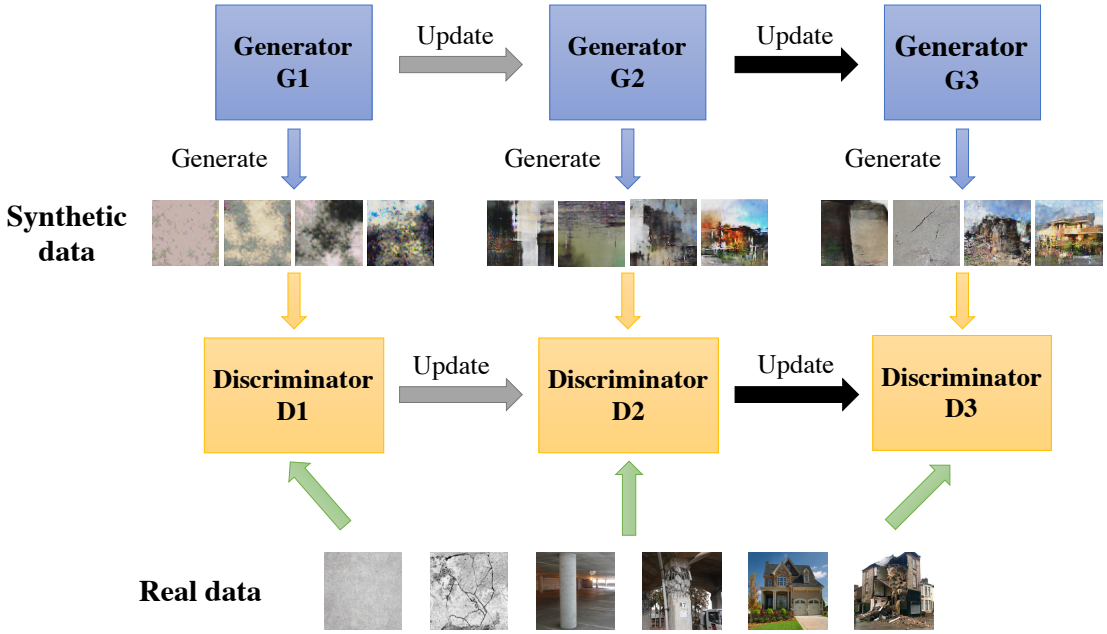


Figure 10.1: Procedure of GAN training

Mathematically, the training procedure of the GAN can be viewed as a game optimization problem, and its optimal solution is the *Nash equilibrium*. However, computing Nash equilibrium has its difficulties, e.g., due to possible non-convex loss functions and high dimensional parameters space. Thus, in practice, using gradient-based algorithms to minimize the loss of the discriminator and the generator simultaneously is preferred.

Denote x as the real data points, z as a high dimensional random variable (also known as the *code*), P_{Data}, P_G, and P_Z as respective distributions of the true data, the generative model, and the random variable, where P_Z is commonly chosen as Gaussian or Uniform. $D(\cdot)$ and $G(\cdot)$ are respectively the discriminator and the generator functions with respective parameters θ_D and θ_G where $D(x)$ is a sigmoid function to output the probability of identifying true data by the discriminator and $G(z)$ produces the synthetic data according to the distribution.

From Goodfellow et al. [39], construct the value function $V(D,G) = V(\theta_D, \theta_G)$ as shown below in Eq. 10.1, where $E_{a\sim P_B}$ denotes the expectation of the data or the random variable a sampled from the distribution P_B, and log is the natural logarithm function:

$$\begin{aligned} V\left(D,\ G\right) &= E_{x\sim P_{Data}}\left[\log\left(D\left(x\right)\right)\right] + E_{x\sim P_G}\left[\log\left(1-D\left(x\right)\right)\right] \\ &= E_{x\sim P_{Data}}\left[\log\left(D\left(x\right)\right)\right] + E_{z\sim P_Z}\left[\log\left(1-D\left(G\left(z\right)\right)\right)\right] \end{aligned} \tag{10.1}$$

The discriminator aims to distinguish whether the fed data are real or synthetic. Thus, it is a binary classification problem and the *empirical cross-entropy* is computed by discrete samples and adopted as the loss as follows:

$$L_D = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\left(D\left(x_i\right)\right) - \frac{1}{N}\sum_{i=1}^{N}(1-y_i)\log\left(1-D\left(x_i\right)\right) \tag{10.2}$$

Labeling the real, x_i, and the synthetic, $\widetilde{x_i}$, data as respectively 1 and 0, i.e., the label $y_i \in \{0,1\}$, then Eq. 10.2 is simplified as shown in Eq. 10.3. In this equation, the empirical quantity of the loss approximately equals its theoretical value when the number of samples N approaches infinity, indicating that optimizing the discriminator by estimating its parameters, $\hat{\theta}_D$, is the same as minimizing the negative of the value function $-V(D,G)$ or maximizing the value function itself as shown in Eq. 10.4. Accordingly, the game maximizes the gain of the discriminator by distinguishing the difference between P_{Data} and P_G as much as possible.

$$\begin{aligned} L_D &= -\frac{1}{N}\sum_{i=1}^{N_{real}} \log\left(D\left(x_i\right)\right) - \frac{1}{N}\sum_{i=1}^{N_{synthetic}} \log\left(1-D\left(\widetilde{x_i}\right)\right) \\ &= -E_{x\sim P_{Data}}\left[\log(D(x))\right] - E_{x\sim P_G}\left[\log(1-D(x))\right] \end{aligned} \tag{10.3}$$

where the number of real and synthetic samples are N_{real} and $N_{synthetic}$, respectively, and they are typically taken such that $N = N_{real} + N_{synthetic}$.

$$\hat{\theta}_D = \underset{\theta_D}{\operatorname{argmin}}\, L_D = \underset{D}{\operatorname{argmax}}\, V(D,G) \tag{10.4}$$

The generator generates the synthetic data where optimizing it aims to generate high quality synthetic data to fool the discriminator. Since it is a Minimax game [27], the generator tries to minimize the gain (i.e., maximizing the value function $V(D,G)$) of the discriminator, defined as the generator loss function L_G to eventually estimate its parameters, $\hat{\theta}_G$:

$$L_G = \max_D V(D,G) \tag{10.5}$$

$$\hat{\theta}_G = \underset{\theta_G}{\operatorname{argmin}} L_G = \underset{G}{\operatorname{argmin}} \max_D V(D, G) \tag{10.6}$$

The well-defined discriminator and generator loss functions above define a Minimax game which aims to optimize them iteratively until both converge to some local minima with the estimated pair parameter sets $(\hat{\theta}_D, \hat{\theta}_G)$. In practice, it is difficult to detect whether they have achieved local minima. Thus, setting a fixed number of iterations is commonly used. The training procedure used in this dissertation for this GAN study is summarized as follows:

Step 0: Initialize the discriminator D and the generator G.

Step 1: Sample N real data points $\{x_1, x_2, \ldots, x_N\}$ using the distribution P_{Data}.

Step 2: Sample N noise sets $\{z_1, z_2, \ldots, z_N\}$ from a selected distribution P_Z.

Step 3: Update D by gradient descent to minimize L_D, where η is the learning rate:

$$\hat{\theta}_D \leftarrow \hat{\theta}_D - \eta \cdot \nabla_{\theta_D} \left(-\frac{1}{N} \sum_{i=1}^{N} \log\left(D\left(x_i\right)\right) - \frac{1}{N} \sum_{i=1}^{N} \log\left(1 - D\left(G\left(z_i\right)\right)\right) \right) \tag{10.7}$$

Step 4: Update G by gradient descent to minimize L_G, using the same learning rate η:

$$\hat{\theta}_G \leftarrow \hat{\theta}_G - \eta \cdot \nabla_{\theta_G} \left(\frac{1}{N} \sum_{i=1}^{N} \log\left(D\left(x_i\right)\right) + \frac{1}{N} \sum_{i=1}^{N} \log\left(1 - D\left(G\left(z_i\right)\right)\right) \right) \tag{10.8}$$

Repeat steps (1) to (4) until convergence is achieved or the designated number of iterations is reached.

In the original GAN [39], the discriminator is updated k times in each iteration before updating the generator. Subsequently, several studies [37, 97] showed a more stable performance by just updating the discriminator once instead of k times. In this study, updating the discriminator only once before updating the generator was adopted. It is noted that the term $\frac{1}{N} \sum_{i=1}^{N} \log\left(D\left(x_i\right)\right)$ in Eq. 10.8 is irrelevant in updating the parameters θ_G since it is 0 when taking gradient with respect to θ_G. However, instead of ignoring this term as stated in [37, 97], it was kept in this study. Because (L_D and L_G share the same expression but with different sign, keeping it is convenient to the coding implementation.

10.2 Deep Convolutional GAN (DCGAN)

In the early studies of GAN [39], a multi-layer perceptron (MLP) was used as an estimator function for the $D(\cdot)$ and $G(\cdot)$. However, this approach was criticized for its unstable performance [39, 97]. Therefore, several new GANs were developed to overcome this concern

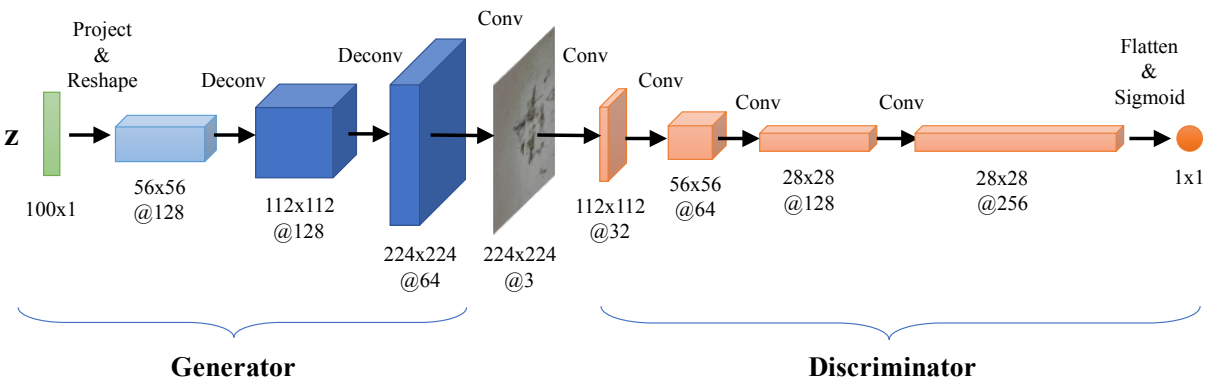


Figure 10.2: Illustration of the DCGAN architecture

by adopting CNN, e.g., Deep Convolutional GAN (DCGAN) [97]. Even though there are many types of GANs to choose from, as an early study in investigating GAN in vision-based SHM, only DCGAN is considered in this dissertation. One example of DCGAN architecture is shown in Fig. 10.2, which is used and described in later experiments.

The DCGAN is not the first GAN combined with deep CNN. Instead, it is defined as using a specific class of CNN architecture for both the generator and the discriminator and restricted to some constraints with the following guidelines [97]:

- Apply batch normalization (BatchNorm) [47] for both discriminator and generator.
- Use Conv operation to replace the deterministic spatial pooling. Specifically, apply deconvolution (Deconv) (i.e., fractional-strided or transpose convolution[1]) in the generator for up-sampling.
- Use Leaky ReLU, Eq. 3.3, as an activation function in the discriminator, except for the output layer which uses the Sigmoid function.
- Use ReLU, Eq. 3.2, as an activation function in the generator, except for the output layer which uses the Tanh function, Eq. 3.4.
- Eliminate the fc-layers after the last Conv layer.

Radford et al. [97] conducted extensive experiments of DCGAN on several datasets including Large-scale Scene Understanding (LSUN) [139], ImageNet-1k[2] [24], MNIST [60], and Faces dataset [97]. The experimental results indicate that the above architectural constraints, i.e., using BatchNorm, Deconv, etc., can contribute to the stable training process across a wide range of datasets and to better synthetic image quality.

[1]It is the reconstruction of a reduced image's spatial resolution to a larger format followed by performing the convolution.

[2]It is a labeled subset of the ImageNet, which has 1000 categories used for the image classification task.

10.3 Evaluation Metrics

GAN, as a DL approach, is empirically evaluated where meaningful metrics are essential. Subjective human evaluation, through the visual quality of the synthetic images compared to the real ones, is a common metric for the GAN performance evaluation. However, the lack of uniform criteria and the time-consuming processing of a large number of images in the human evaluation procedure greatly reduce the efficiency in augmented synthetic data selection and also impede the development of automated end-to-end data augmentation frameworks for real time applications. Thus, developing efficient and quantitative metrics is an active research area of GAN.

Inception Score

Inception Score (IS), as an alternative to human evaluation, was first proposed in [102]. The intuition behind the IS is explained by the fact that a well-trained DL model which achieves high classification accuracy can be well-correlated with the human judgement of the image quality, i.e., higher IS implies better visual quality of the generated image. The term "Inception" comes from using a pre-trained InceptionNet [122] model as the classifier in the original study [102].

For a set of synthetic images, x, IS is computed as follows:

1. Use a pre-trained Inception model to determine the conditional probabilities $p(y|x)$ of the class labels assigned by the network y to each of image x.
2. Compute the empirical marginal class distribution $\hat{p}(y)$ (Eq. 10.9) as the estimate of the true distribution $p(y)$.
3. Compute the IS using an approximate exponential of the Kullback–Leibler (KL) Divergence (Eq. 2.18) between $p(y|x)$ and $\hat{p}(y)$, Eq. 10.10.

$$\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_i) \tag{10.9}$$

$$\begin{aligned} IS(G) &= \exp\left(E_{x\sim P_G} D_{KL}\left(p(y|x) \parallel p(y)\right)\right) \\ &\approx \exp\left(\frac{1}{N}\sum_{i=1}^{N} D_{KL}\left(p(y|x_i) \parallel \hat{p}(y)\right)\right) \end{aligned} \tag{10.10}$$

If an image is real and meaningful, the objects and major targets in the image should be clear and sharp making it easily identifiable, i.e., low cross-entropy of $p(y|x)$. Moreover, realistic images should indicate high diversity of different object categories, i.e., high entropy of $p(y)$. According to [6], the expected KL Divergence (Eq. 10.10) can be expressed as:

$$IS(G) = \exp(H(y) - H(y|x)) \tag{10.11}$$

where $H(y)$ and $H(y|x)$ are the entropy of $p(y)$ and cross-entropy of $p(y|x)$, respectively. This simultaneously encodes the two conditions in this paragraph, and it further proves that images will be more meaningful with higher IS values.

The IS faces some limitations and impediments in real practice. In the original work [102], the classifier InceptionNet was pre-trained on ImageNet [24] (denoted as reference dataset), and synthetic images were learned from CIFAR-10 [57]. Both object categories and tasks of CIFAR-10 nearly belong to a subset of ImageNet. It was expected that the classifier trained on the ImageNet has high recognition ability on the CIFAR-10 data. However, in the domain of vision-based SHM, both structural images with damaged patterns and detection tasks of scene level or damage state classification significantly differ from those in the ImageNet, where the recognition ability of the pre-trained classifier can be far from that of a human expert. Thus, another score should be proposed for the study related to the vision-based SHM.

Self-Inception Score

As mentioned above, the principle that pre-trained InceptionNet classifier can replace human judgement is due to its high classification accuracy. Therefore, any other well-trained classifier with higher accuracy can be an alternative for the InceptionNet. Unlike the original IS using ImageNet for training classifier, if the dataset used for training GAN is well-labeled and one classifier can obtain an acceptable accuracy on this dataset, then the classifier itself can be assumed close to human judgement but possibly slightly weaker. In other words, the same data are used for training both the classifier and the GAN in recognizing itself and generating synthetics images. Therefore, the IS computed under such pre-trained classifier is called *Self-Inception Score* (*SIS*). This SIS can be thought of as a reasonable metric to overcome the drawback due to lack of a large reference dataset.

In Eq. 10.9, the marginal class distribution is estimated by empirical distribution where its value may vary with the dataset size. Previous studies [6, 102] divided the large number of synthetic images into groups with different group sizes. However, they only studied the influence of group sizes without considering a new IS formulation. Here, a *split plan* is defined as the one that splits the image set into n_{split} groups. This is followed by determining the mean and standard deviation (Std) of IS values from all the split groups. Usually, lower mean of the IS and higher Std are observed with more group, where the Std value is usually used to evaluate the variance of each group. These issues are considered in SIS through averaging the IS values according to different split plans. Thus, SIS considers local effects induced by the partial synthetic images, which are absent if computing the IS from all the data. The procedure of the SIS computation is listed below, and steps (4) to (6) are illustrated in Fig. 10.3.

1. Partition the raw dataset into training set and validation set.

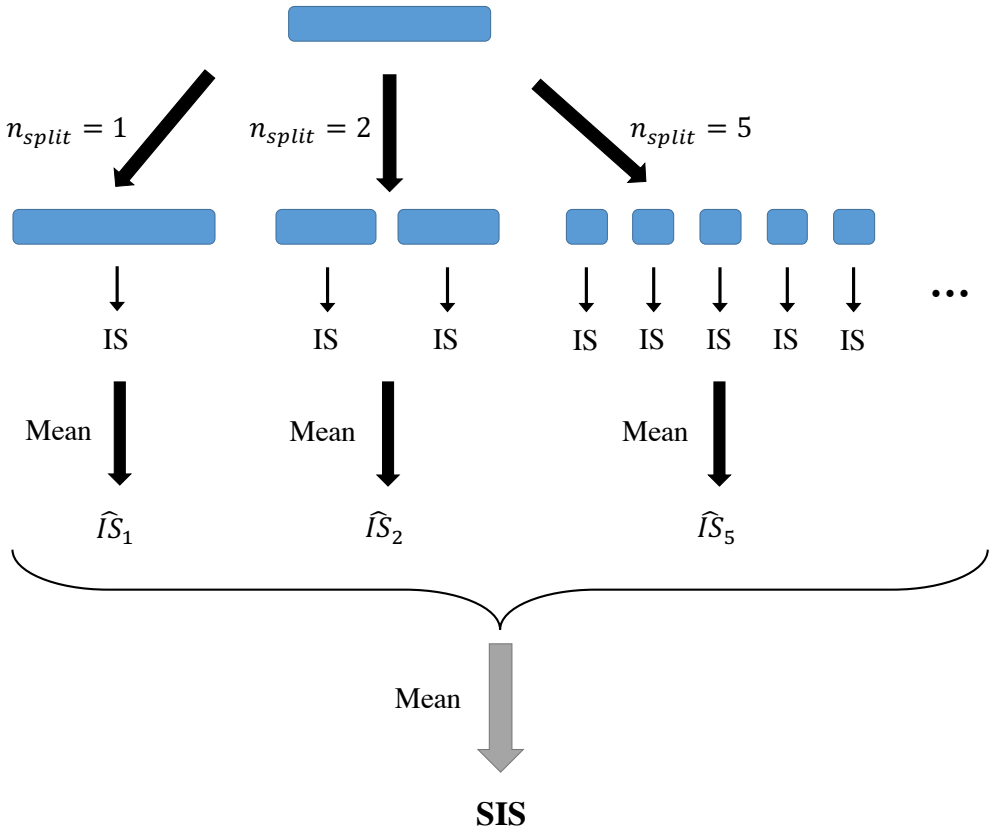


Figure 10.3: Split plan for computing the SIS

2. Train the classifier with the training set by multiple deep neural networks, e.g., VGGNet [113], InceptionNet [122], and ResNet [42], and pick the classifier with the best performance on validation set.

3. Train the GAN with the training set only and generate a large number of synthetic images.

4. Design split plans $n_{split} = \{1, 2, 5, \dots\}$.

5. Compute the mean values of the different IS from each split plan configuration to produce $\widehat{IS}_i$ ($i \in n_{split}$), refer to Fig. 10.3.

6. Compute the mean and the Std of the $\widehat{IS}_i$ values in step (5) where the mean value corresponds to the required SIS, refer to Fig. 10.3, and Std measures the variation of the mean of IS among the split groups.

Although the SIS mitigates the limitations of the IS and considers the local effects, it is still a rough score. Similar to the IS, the SIS is sensitive to small weight changes of

the network and synthetic images with high SIS may be unrealistic as observed in some adversarial samples [6].

Generalization Ability

The SIS considers the evaluation in terms of information gain and loss, which is limited, i.e., the image may be given a high IS value, even if it is classified with the wrong label. Thus, the *Generalization Ability* (GA) is another independent metric to evaluate whether these synthetic images are meaningful enough to be recognized by the classifier. It is inspired by the metric of overall accuracy (Eq. 2.7) and the discriminator part of the GAN, which distinguishes the synthetic images from the real ones to improve the performance of the generator. Different from regular setting with both training and validation from real data, GA measures the accuracy achieved by the classifier on the real validation set but such classifier is only previously trained on the synthetic images generated by the GAN. For example, consider a task to classify object level images with damaged and undamaged states. Herein, synthetic images are generated and labeled firstly, and then they are used to train a strong classifier, e.g., VGGNet or InceptionNet. Finally, the classifier is tested using a validation set with real images only to quantitatively determine the quality, i.e., the overall accuracy (Eq. 2.7), of the synthetic images. If the generated data are realistic, they will have similar representations, e.g., color and visual contents coded and stored digitally, and essential features to those of the real data, leading to a high validation accuracy on the validation set. Moreover, from a statistical point of view, GA can be thought of as a measure of the distribution difference between the synthetic training set and the real validation set in some sense. Thus, GA is another reasonable and quantitative metric to evaluate the GANs.

Chapter 11

GAN for Structural Image Data Augmentation & Classification

In Chapter 10, several basics of the Generative Adversarial Network (GAN) were briefly introduced. In this chapter, two major topics are discussed, namely performance of using GAN in generating structural images, and the potential usage of GAN-based image augmentation method in classification problems under low-data regimes.

11.1 Leaf-Bootstrapping Augmentation

Compared to several datasets used in [97], where images share similarities with few variations, images in the ϕ-Net vary significantly due to different scales, irregularity of damage patterns, complex structural designs, and noisy backgrounds. Moreover, the size of the dataset is still small. These drawbacks may limit the GAN performance using the ϕ-Net dataset. Preliminary experiments were conducted by feeding all training images into one well-designed deep convolutional GAN (DCGAN) model and over 25,000 synthetic images were produced by a fine-tuned GAN. As expected, the quality of images generated by DCGAN, which was directly trained with all training images, was not satisfactory. Although there existed several realistic synthetic images, several of other ones looked like those in Fig. 11.1, which were mixtures of different levels. For example, the top four images show the sky and part of the building outlines belonging to the structural level and the bottom four images look like a wall surface belonging to the pixel level.

Statistically speaking, as a type of generative model, the ultimate goal of GAN is to learn the distribution of the data. In the case of vision-based SHM, from the perspective of the data distribution, the scene level or the damage state should have a mixed distribution by several sub-distributions with multiple peaks and ranges. For example, from the hierarchy of the ϕ-Net (Fig. 4.14), the structural image labeled as damaged or undamaged state can be further separated and sorted from: (1) the pixel level with damaged state (PD), (2) the object level with damaged state (OD), and (3) the structural level with damaged state (SD)

or from (i) the pixel level with undamaged state (PUD), (ii) the object level with damaged state (OUD), and (iii) the structural level with damaged state (SUD), respectively, Fig. 11.2. Failure of directly training GAN is due to the limited size and complexity of the dataset (without additional information like labels to reduce the complexity) making it difficult for the original GAN to learn the complex mixture distribution (black curve in Fig. 11.2) directly.

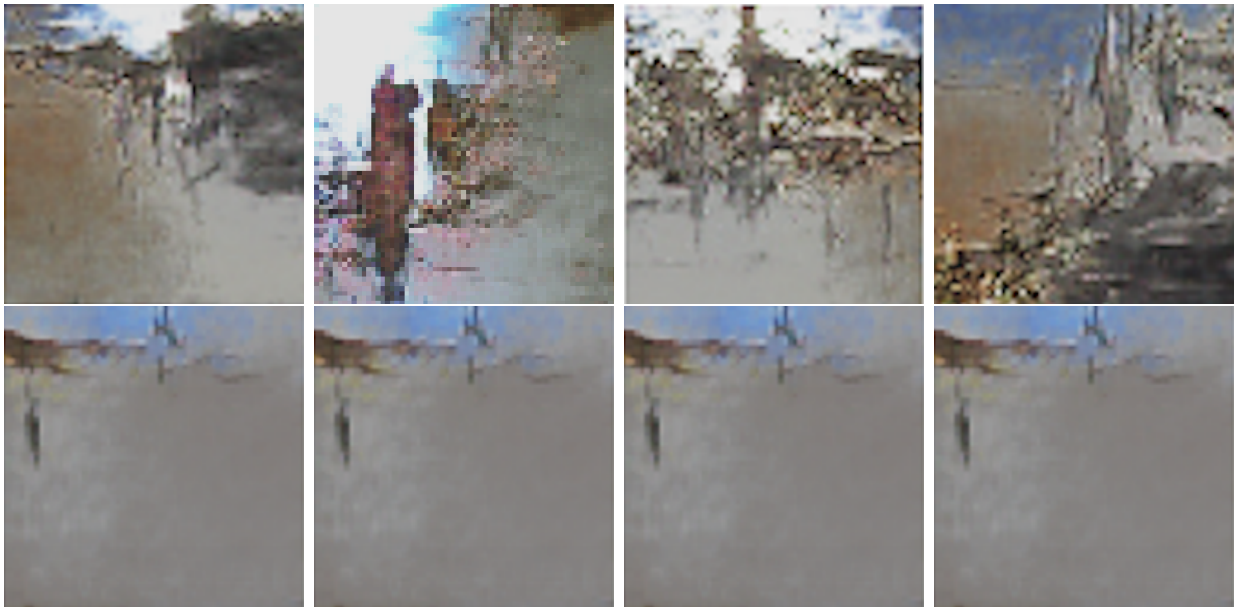


Figure 11.1: Images generated by GAN trained with all the training images

Based on domain knowledge and Bayesian theorem [9, 66], the distribution (probability density function) of the structural images can be computed from the combination of distributions of sub-categories, i.e., density of PD, OD, and SD or that of PUD, OUD, and SUD. Comparing to learning a complex distribution directly, it is much easier to start from learning its sub-distributions and then learn the aggregated distribution, which shares similar idea with the Gaussian Mixture Models [9, 66]. Inspired by this concept and combining it with the hierarchy ϕ-Net framework, a new training method for GAN is proposed, namely Leaf-Bootstrapping (LB). Its name originates from the purpose of making use of the leaf node distribution and generating data from a well-learned distribution, where the latter part also shares the similar idea of bootstrapping [84] that samples data from a learned distribution.

In most application scenarios, using the whole ϕ-Net framework is time-consuming and inefficient. Instead, the simplified hierarchy ϕ-Net using shallow representations, Fig. 4.15, can be considered, and these shallow representations are expected to work well with the LB method for GAN training. For example, while adopting depth-2 representation (Fig. 4.15), instead of generating synthetic images for the damaged state or the scene level directly, GAN models now can be trained on their sub-classes from the leaf nodes. Accordingly, six GAN models are trained for PUD, PD, OUD, OD, SUD, and SD. Based on each sub-class, the class-specified synthetic images are produced, e.g., synthetic PD images are sampled from the

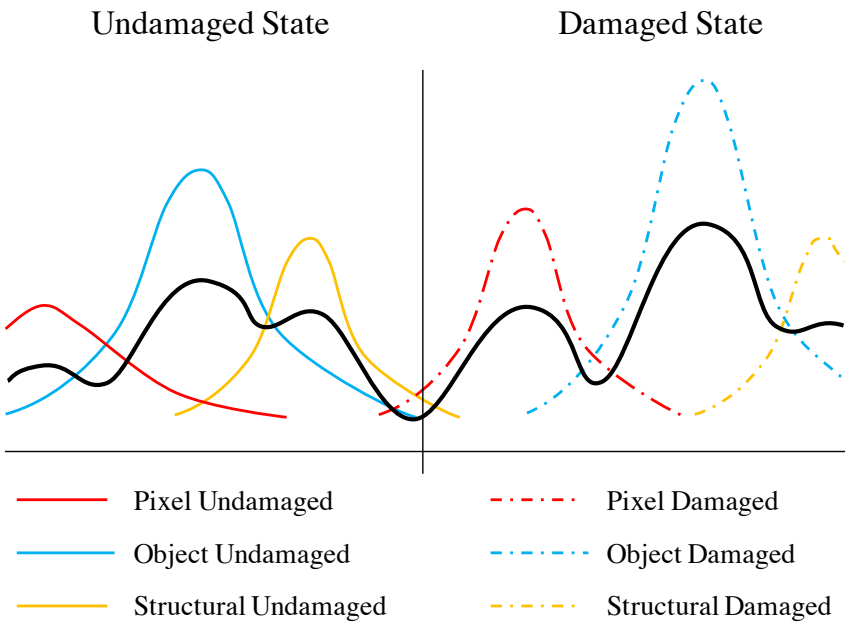


Figure 11.2: Diagram of a mixture distribution for the structural images in an abstract dimensional space. (Note: horizontal axis represents high dimensional features and vertical axis represents the magnitude of the probability density function)

GAN trained on the PD images. Then, the synthetic images are reassembled according to their root tasks, e.g., for the synthetic pixel level images used in the scene level task, synthetic images of PD and PUD are mixed, Fig. 11.3. Similarly, in the damage state task, synthetic damaged state images are mixed from synthetic PD, OD, and SD. Moreover, LB method can be easily extended to depth-3 framework (Fig. 4.15), which requires mixing deeper nodes prior to their parent nodes, i.e., for access to PD, synthetic images of non-spalling and spalling need to be assembled first. In summary, the LB method through using simplified depth-2 or depth-3 framework benefits the GANs training and samples images of higher quality through minimizing the inner class variations by simply pre-clustering based on domain knowledge.

As mentioned above, since the direct usage of the GAN did not work, in the following experiments with respect to structural image generation, the LB method is adopted in the GAN training. It should be noted that, in this dissertation, the GAN model using LB method is denoted as LB-GAN for short.

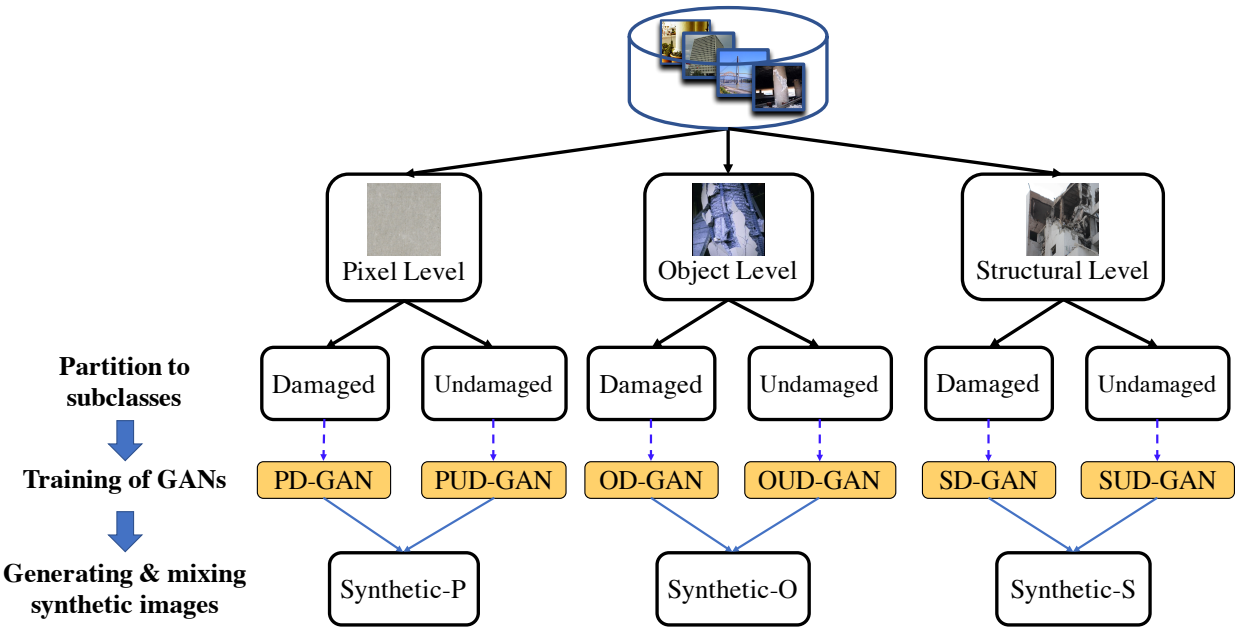


Figure 11.3: Leaf-Bootstrapping for synthetic scene level using ϕ-Net depth-2 representation

11.2 Experiments on Synthetic Structural Image Generation

Experimental objectives

As the first study of using GAN in vision-based SHM, in the previous section, it has been shown that simply direct usage of GAN does not work for generating reasonable structural images. Thus, it is important to determine whether GAN can effectively work in vision-based SHM when using an appropriate training method.

In the experiments of synthetic image generation, two common scenarios were considered, scene level identification and damage state check, which are actually two basic ϕ-Net benchmark classification tasks. Moreover, besides simply validating the effeteness of the GAN performance using LB method, several specific questions are explored as follows:

1. Does GAN using LB method with depth-2 representation work better than the directly trained one?

2. Does a cleaner but smaller scale dataset generate synthetic images of better image quality?

3. Can a convention data augmentation (DA) method further improve the GAN performance?

4. Can deeper tree representation such as depth-3 outperform the shallower ones such as depth-2?

For answering each question, the same experimental procedure and evaluation technique are conducted. The considered evaluation metrics are: (1) synthetic image quality by human evaluation, (2) GAN performance by the Self-Inception Score (SIS), and (3) GAN generalization ability (GA).

Experimental setup

ϕ-Net-GAN dataset

All labeled structural images are directly collected from the ϕ-Net dataset. Because GAN is known to be inferior due to its unstable training procedure [39, 97] and since this is the first study to apply GAN in a non-benchmark dataset, a rigorous image selection procedure and distribution design were conducted as data preprocessing. Thus, all raw data were well-selected from the dataset with multiple labels of the scene level and the damage state with relatively high quality and less ambiguity. Subsequently, the images were rescaled to size 224×224. Finally, 6,821 images, in total, were used as the entire experimental dataset for this study, and this dataset is named ϕ-Net-GAN. Reasonably balanced label ratios for both the scene level and the damage state were achieved as shown in Fig. 11.4.

A split ratio 4:1 was adopted to separate the training and the test datasets, and similar label distributions were kept in both datasets. The numbers of data for both datasets are listed in Table 11.1, which is useful for later classification experiments. It is noted that only data in the training dataset were used in training the GAN model.

Table 11.1: Numbers of raw data in training and test datasets for six categories

Dataset	**PUD**	**PD**	**OUD**	**OD**	**SUD**	**SD**
Training	865	1,096	555	1,107	992	771
Test	218	278	137	286	247	193

In order to address the above mentioned four questions, the following four training datasets were designed considering two scenarios:

1. D2-All: All raw training images with depth-2 representation.

2. D2-Clean: Select up to 128 images per class from D2-All with less noise for PD, PUD, OD, OUD, SD, and SUD.

3. D2-Clean&Aug: Augment D2-Clean up to 1,024 images per class using a conventional DA method.

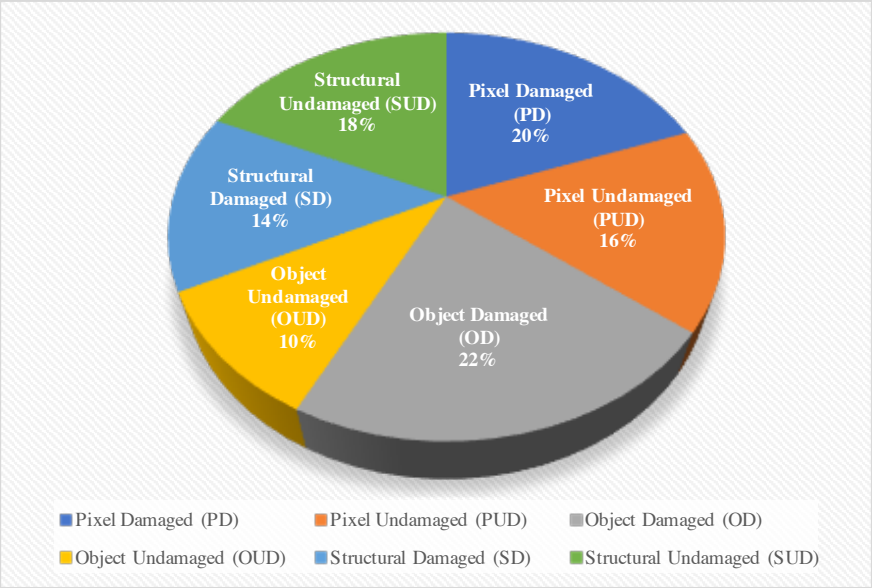


Figure 11.4: Label ratio of the ϕ-Net-GAN

4. D3-Clean&Aug: Select up to 64 and 128 images per class for the respective depth-3 and depth-2 representations according to the configuration in Fig. 4.15, then perform DA up to 1,024 images per class for PD, PUD, OD, OUD, SD, and SUD.

It is noted that while in image selection of D3-Clean&Aug using depth-3 representation, considering the complexity of the visual patterns in the object level, the images were split into "beam/column" and "wall" for both undamaged and damaged states, as an extension of the representation shown in Fig. 4.15.

Network configuration

The size of the generated synthetic images from the original DCGAN is equal to or smaller than 64×64, which is much lower than expected. It is pointed out that in the TL experiments in Chapter 5, several deep CNN models, e.g., VGG-type and ResNet, achieved stable and relatively optimal performance in structural damage detection using 224×224 as the network input size, where the feeding images were simultaneously rescaled to match the same size. Therefore, a specific DCGAN configuration using 224×224 as the network input size for the discriminator and output size for the generator was designed for this experiment. In other words, the synthetic image size in this experiment is 224×224, which is an improvement over the previous work [97]. Following the constraints of the DCGAN in Section 10.2, the configurations of the discriminator and the generator are shown in Table 11.2, Table 11.3, and Fig. 10.2.

Table 11.2: Network configuration of DCGAN generator (N: # data fed into the network)

Layer	**Filter Size (#)**	**Activation**	**Shape**	**Notes**
Input	-	-	(N, 100, 1)	Noise generated from Normal distribution
Fc-layer	-	ReLU	(N, 401,408)	401,408 = $56 \times 56 \times 128$
Reshape	-	-	(N, 56, 56, 128)	-
Batch Norm	-	-	(N, 56, 56, 128)	Momentum = 0.8
Deconv	3×3 (128)	ReLU	(N,112, 112, 128)	Up-sampling
Batch Norm	-	-	(N, 112, 112, 128)	Momentum = 0.8
Deconv	3×3 (64)	ReLU	(N, 224, 224, 64)	Up-sampling
Batch Norm	-	-	(N, 224, 224, 64)	Momentum = 0.8
Conv	3×3 (3)	Tanh	(N, 224, 224, 3)	-

Table 11.3: Network configuration of DCGAN discriminator (N: # data fed into the network)

Layer	**Filter Size (#)**	**Activation**	**Shape**	**Notes (α: −ive slope coef. in Leaky ReLU)**
Input	-	-	(N, 224, 224, 3)	-
Conv	3×3 (32)	Leaky ReLU	(N, 112, 112, 32)	Conv stride = 2; $\alpha = 0.2$
Dropout	-	-	(N, 112, 112, 32)	Dropout rate = 0.25
Conv	3×3 (64)	Leaky ReLU	(N, 56, 56, 64)	Conv stride = 2; $\alpha = 0.2$
Dropout	-	-	(N, 56, 56, 64)	Dropout rate = 0.25
Batch Norm	-	-	(N, 56, 56, 64)	Momentum = 0.8
Conv	3×3 (128)	Leaky ReLU	(N, 28, 28, 128)	Conv stride = 2; $\alpha = 0.2$
Dropout	-	-	(N, 28, 28, 128)	Dropout rate = 0.25
Batch Norm	-	-	(N, 28, 28, 128)	Momentum = 0.8
Conv	3×3 (256)	Leaky ReLU	(N, 28, 28, 256)	Conv stride = 1; $\alpha = 0.2$
Dropout	-	-	(N, 28, 28, 256)	Dropout rate = 0.25
Flatten	-	-	(N, 200,704)	200,704 = $28 \times 28 \times 256$
Fc-layer	-	Sigmoid	(N, 1)	-

Training configuration

The common issue in GAN training is that hyper-parameters for DCGAN are quite sensitive. After extensive fine-tuning operations, it is found that using Adam optimization [53] with learning rate 0.0001 or 0.00005, $\beta_1 = 0.5$, and $\beta_2 = 0.999$ was stable and achieved good image quality. Moreover, for the considered non-benchmark dataset, it is difficult to obtain Nash equilibrium. Thus, in this study, 40,000 maximum training iterations were considered and the best GAN model was selected with the best image quality by human expert judgement. Since the LB method was adopted, multiple GAN models were trained and most GAN models were found to achieve the best performance at about 25,000 iterations. Similar to the training procedure of the original GAN (Section 10.1), the computational cost for each iteration including the gradient updating steps (Eqs 10.7 & 10.8) depends on the batch size $2N$. In the experiments, the batch size of 16 was used and corresponded to an average time of 0.12

sec. for each iteration in the GAN training procedure. In other words, for the previously mentioned maximum of 40,000 iterations, approximate training time for one GAN model was about one hour and 20 minutes. As for the synthetic image generating time, an average of only 0.003 sec. per image (without saving operation) was achieved.

As suggested in [102], IS becomes suitable when using a large number of samples (i.e., at least 50,000 total and 5,000 for a single class). The considered two scenarios were classification problems with few classes and for computational efficiency, 5,000 images per class for the six leaf nodes (PD, PUD, OD, OUD, SD, and SUD) were generated to form a set of 30,000 synthetic images. For depth-3 representation, 2,500 images per class were generated for sub-classes of damaged and undamaged states ("SP & NSP for PD", "Beam/Column & Wall for both OD & OUD", and "Non-collapse & Global collapse for SD"), which also summed up to 30,000 images together with the 5,000 images per the three classes of no damage, i.e., PUD, OUD, and SUD. In order to satisfy at least 5,000 images for each class, only three split plans (n_{split}=$\{1,2,5\}$) were adopted. In other words, for a total of 30,000 synthetic images, 30,000, 15,000, and 6,000 were the size of each split group according to these three split plans.

Human evaluation

After extensive experiments and reassembling back to the six leaf tasks in depth-2 representation, several samples of the synthetic images are shown in Fig. 11.5 for four different cases[1]. Starting from Synthetic-D2-All, it is shown that the GAN already learned some basic patterns such as cracks, spalling, shape of column/wall, etc., but the images look blurry and lack details. Compared to the results of each leaf node, the image quality and training difficulties vary from one to another, especially in the object level. Even though the outlines of the structural components can be roughly recognized, most images contain noise, such as green parts in the $1st$ row, and $1st$ & $2nd$ columns of OUD, which make the generated images unrealistic. This can be explained by the fact that the raw object level images contain many irrelevant parts, which confused the GAN. Unlike other examples, synthetic PUD and SUD images are more realistic to the human eyes, especially when evaluated in relatively low resolution. Thus, filtering and de-noising may be necessary.

As for Synthetic-D2-Clean, after data cleaning, the generated images show more details, such as reasonable damage patterns and outlines of the structural component. In the PD examples, cracks are much sharper and clearer than the previous case, and another important observation is that cracks occur in different locations with different slopes, which is an evidence that the GAN can learn well the visual features of the cracks, but it may lack the physical meaning. In the OD and OUD examples, the boundaries and lighting make the structural components appear more in three-dimensional, and the noise level slightly decreases. However, due to significant decrease of the amount of data per class, more challenges are faced during the training procedure, such as losses for both the discriminator and the generator do not change for the first several hundreds of iterations, leading to only generating some

[1]High resolution versions of the synthetic images can be found in `http://stairlab.berkeley.edu/data/`.

noisy images. Therefore, to overcome these issues, multiple trials for training are necessary benefiting from the randomness of the CNN parameters and the input latent variables. Another problem is the *mode collapse* where only few types of images are generated by the generator and these generate images lack variety, and most images are similar with limited changes, such as most images in the PD have a similar texture.

As an augmented version of D2-Clean, GAN trained from D2-Clean&Aug slightly resolved the above issues and achieved some improvements in the image quality, e.g., spalling patterns in PD are more realistic. In addition, both cracking and spalling patterns are observed in OD, and less noise and sharper boundaries are obtained in the structural level. However, undesired shifting patterns are learned by the GAN and observed in the structural level images due to the affine[2] DA. Therefore, translation/shift operations in augmentation should be limited to a smaller range in future studies.

As for Synthetic-D3-Clean&Aug, compared to previous cases, it achieved the best performance. In the pixel level examples, GAN can generate more realistic cracking and spalling patterns benefiting from further splitting into two sub-classes, namely non-spalling (NSP) and spalling (SP). In the object level examples, more complex and detailed structural components can be generated, such as the good quality of single column images ($1st$ row of both OD & OUD), wall with corner ($2nd$ row, $3rd$ column of OD), wall with different colors such as white in top and pink in bottom ($3rd$ row, $1st$ & $2nd$ columns of OD), brick texture of wall ($3rd$ row, $3rd$ column of OUD), and appearance of plants ($3rd$ row, $1st$ column of OUD). In the structural level examples, due to further splitting in the SD node, the global collapse ($1st$ row in SD) and the non-collapse ($3rd$ row in SD) become more distinguishable and sharper outlines can be observed. Because of no further splitting in the SUD node, the synthetic SUD images in this case are similar to those with synthetic D2-Clean&Aug.

In all cases, the synthetic pixel level images describe a small range of a structural component with simple and few patterns, the synthetic object level images show the outlines of structural components with adequate lighting and shadow effects, and the synthetic structural level images present realistic shape of building structures with some details, e.g., windows and doors. Furthermore, for the classification, with the above-mentioned sharp differences, the synthetic images are expected to distinguish between the three different levels in the scene identification. For the damage state check, differences between damaged and undamaged states are obvious. Overall views of the synthetic images in an undamaged state are smooth and aligned, while those of a damaged state are cruder and more disorganized. Especially in SD, debris of damaged structures make the scene chaotic which describes the structural collapse features well. However, while comparing to the real images with 224×224 resolution, the synthetic images in all cases still look somewhat blurry, which is the common issue observed in most GAN applications. This drawback can be attributed to: (1) the averaging effect of GAN, where the objective functions (Eqs 10.2 & 10.2) are expressed in the

[2]In geometry, an affine transformation is a function which maps an affine space onto itself while preserving both the dimension of any affine subspaces and the ratios of the lengths of parallel line segments. Examples of affine transformations include translation, scaling, similarity, reflection, rotation, shear mapping, and compositions of them in any combination and sequence.

sense of *expectations*, and (2) using a relatively large reconstruction image size with only a small amount of training data. Moreover, all synthetic images still lack some natural details, such as dirt, decoration, etc. This is particularly the case for PUD where all synthetic images look like a pure color patch. However, this issue does not harm the classification towards the damage state check, which is validated and discussed in the classification experiments in a later section.

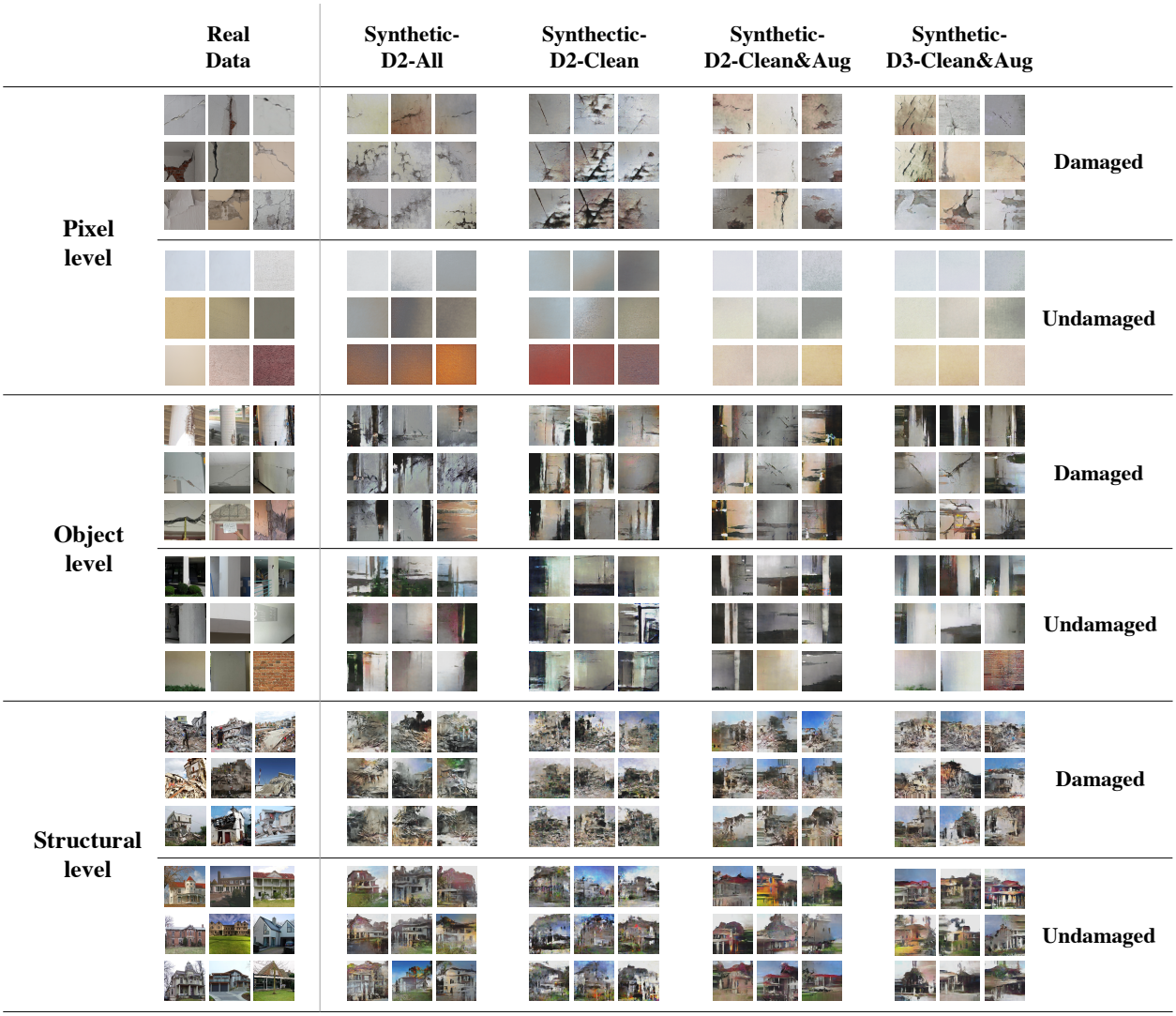


Figure 11.5: Real images vs synthetic images generated from four cases. (Note: Synthetic-D2-All means synthetic images generated by the GAN trained from D2-All dataset, and the other three columns of synthetic images follow a similar nomenclature.)

Scenario 1: Scene level identification

Self-Inception Score

In the scene level identification, 30,000 synthetic images were generated with 10,000 per class for P, O and S. To calculate the SIS, a strong classifier (VGG-19) was pre-trained on the real training images with about 98% validation accuracy, which is expected to have strong recognition properties close to the human judgement or even better. Therefore, the SIS for the whole real training images (Real-All where each class has about 1,000 real images) is considered as the upper bound instead of the higher theoretical bound (computed as the number of classes, i.e., 3 in this scenario refer to [6]). This is a compromise due to the lack of a large reference dataset. However, the lower bound is the same as the IS, i.e. 0 [6]. From Table 11.4, it is observed that SIS for the Real-All is the highest with relatively small standard deviation (Std), consistent with the expectations.

To explore the GAN performance, the synthetic images were generated based on different settings. The Synthetic-D2-All represents the synthetic images generated from the GAN learned from Real-All dataset. Its SIS is relatively high, which has only 10% reduction compared to the real images, Table 11.4. These results demonstrate that the synthetic images are of good quality in terms of the scene level. However, there are some images that share features from two scene levels, P and O, making it difficult to classify them accurately. Thus, a high cross-entropy of $p(y|x)$ maybe achieved. To explore this issue, the cleaned dataset (D2-Clean) was used to generate synthetic images (Synthetic-D2-Clean) leading to a lower SIS compared to Synthetic-D2-All. Two reasons can explain this observation: (1) the cleaned dataset is too small, which makes it difficult for the GAN to capture the precise distribution, and (2) the reduced noise by the cleaning operation leads to the decrease in the cross-entropy of $p(y|x)$ which increases the SIS. However, this increase cannot compensate for the decrease of the entropy of $p(y)$ due to losing data variety by ruling out a certain number of data. In other words, according to Eq. 10.11, the decrease of the entropy of $p(y)$ dominates the decrease of the cross-entropy of $p(y|x)$, which causes the final decrease of the SIS for the Synthetic-D2-Clean.

To address the Synthetic-D2-Clean problem, conventional affine DA was employed to increase the dataset from 128 per class to 1,024 per class leading to the increase of the SIS from 1.817 to 1.879 for Synthetic-D2-Clean&Aug, which is even higher than that for the Synthetic-D2-All (1.840). Moreover, under the setting of Synthetic-D3-Clean&Aug, the SIS (1.903) improved further, benefiting from the clearer and higher quality images even though it scarifies some complex and noisy images during cleaning. Thus, the LB method with DA achieved higher SIS in the synthetic settings. Moreover, comparing the SIS between its upper bound (2.063) and the case of Synthetic-D3-Clean&Aug, the gap is only around 0.16 (only 7.76% of the upper bound), which is relatively small considering the range and the Std of SIS. This result demonstrates the effectiveness of the proposed method in the scenario of the 3-class scene level classification.

Generalization Ability

Using the real training dataset, 97.7% validation accuracy was achieved, which is the upper

Table 11.4: SIS and GA for synthetic images in the scene level identification

Dataset	Real-All	Synthetic-D2-All	Synthetic-D2-Clean	Synthetic-D2-Clean&Aug	Synthetic-D3-Clean&Aug
$SIS \pm$ Std	2.063 ± 0.149	1.840 ± 0.164	1.817 ± 0.175	1.879 ± 0.163	1.903 ± 0.164
GA	97.70%	85.14%	84.92%	82.41%	83.52%

bound of GA. Similar settings were considered for the synthetic images. From Table 11.4, Synthetic-D2-All has the highest validation accuracy using the synthetic training dataset because it remains the most informative for the distribution including complex and noisy data, making the generated synthetic images the most generalized ones. Since the cleaned dataset for Synthetic-D2-Clean loses some variety, the accuracy (84.92%) is slightly lower compared to that of the Synthetic-D2-All (85.14%). Similarly, when going deeper for depth-3 representation, the dataset loses more information. Thus, Synthetic-D3-Clean&Aug has even lower accuracy (83.52%). One interesting observation is that applying DA in training, Synthetic-D2-Clean&Aug accuracy declined by about 2.5% compared to the case without DA. This augmentation on a cleaned dataset puts too much emphasis on the cleaned data, which ignores some complex and abstract patterns from the eliminated images, leading to the loss of generality.

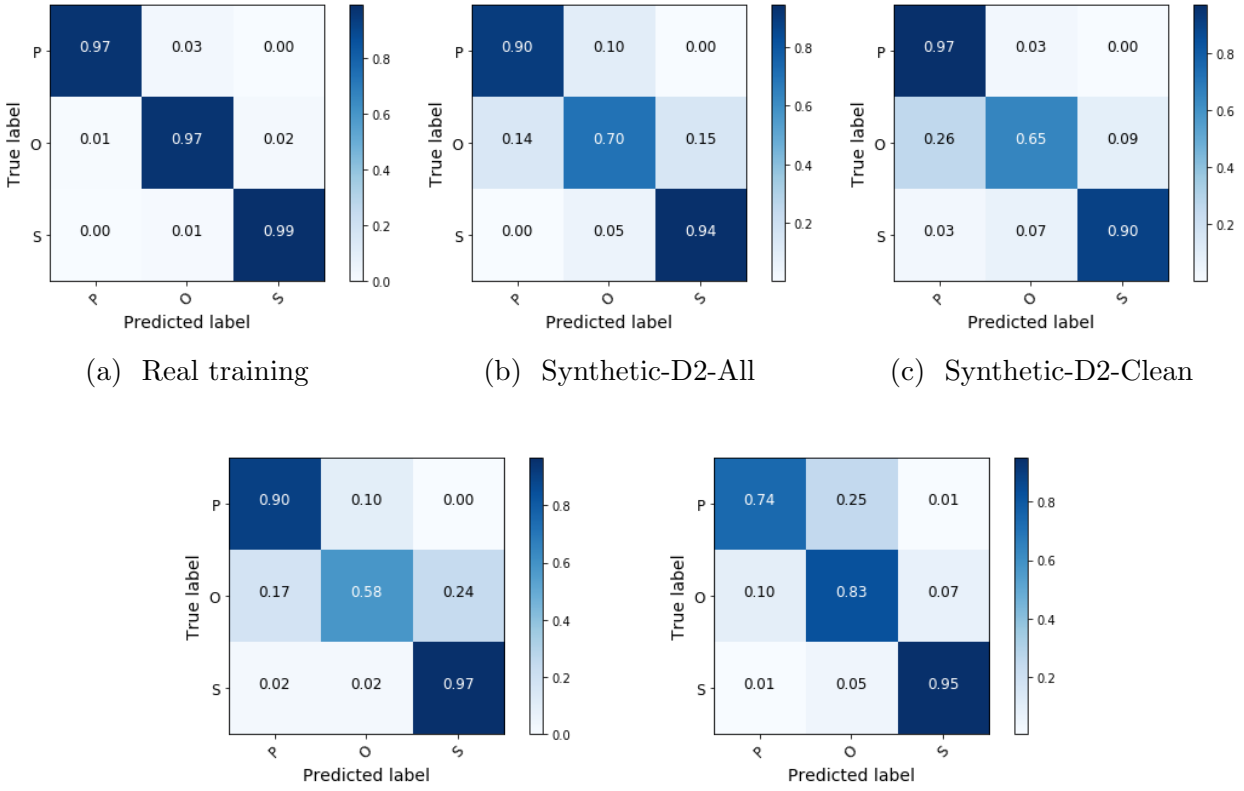
To better explore the validation accuracy, the confusion matrices are evaluated, Fig. 11.6. From Figs 11.6b to 11.6d, it is observed that the classifier predicted many object level images as non-object labels (P & S). Because there are several sub-classes in the object level, e.g., wall, column and beam, and when generating synthetic images only using depth-2 representation, GAN merged the features from different sub-classes resulting in a larger difference between the real and the synthetic images. While sing depth-3 representation (Fig. 11.6e), the wall, column and beam are separated making the object level performance much better than that of depth-2. As a trade-off, when splitting the dataset into more granular classes, some sub-classes in the object level are similar to those in the pixel level (e.g., some wall images look like those in the pixel level). As a result, some pixel level images are mistakenly predicted as label O for a wall in the object level. Moreover, the object level is the intermediate level between pixel and structural levels. Thus, it shares some common features with the other two levels, which may confuse the classifier.

In a broader view, GA values obtained from all the synthetic cases are close and have an evident gap (say around 12% difference) with the GA from the ground truth. Except for the reasons mentioned above, this issue can be attributed to the distribution difference between the assembled set of the synthetic images and that of the real images. Therefore, better GAN model and more reasonable assembling technique should be investigated in future studies for further improvements.

Scenario 2: Damage state check

Self-Inception Score

Similar to the scene level classification, VGG-19 was pre-trained on the whole real training images and achieved 96% validation accuracy. The same dataset setting was used as the



(a) Real training (b) Synthetic-D2-All (c) Synthetic-D2-Clean

(d) Synthetic-D2-Clean&Aug (e) Synthetic-D3-Clean&Aug

Figure 11.6: Confusion matrices of the GA in the scene level identification

previous experiments, and results are shown in Table 11.5. Apparently, Real-All achieved the highest SIS, which represents the upper bound of the score (the theoretical bound is 2 for this scenario with 2 classes). Among the synthetic datasets, results from synthetic-D2-All have the highest SIS without further improvement of the SIS with dataset cleaning, augmentation, or use of deeper representation. Different from the scene level classification where the synthetic images of P, O, and S are quite different from each other, in the present scenario, several noisy areas, e.g., green and blurry regions in the undamaged state images may be mistaken (even by human evaluation) as damaged patterns by the classifier. Due to losing several images, results from the cleaned dataset are inferior to those using all raw data. On the other hand, results from using DA and using deeper representation are better than the data cleaning where their scores are similar to those of Synthetic-D2-All and very close to the upper bound score. Since several compromises were made in computing the upper bound and the pre-trained classifier, the score may not represent the real situation consistent with concerns by [6] where unreasonable adversarial images may have higher scores than realistic ones. Thus, as a metric within the perspective view of information, the performance of Synthetic-D2-All, Synthetic-D2-Clean&Aug and Synthetic-D3-Clean&Aug

cases are considered acceptable under the SIS metric.

Table 11.5: SIS for synthetic images in the damage state check

Dataset	Real-All	Synthetic-D2-All	Synthetic-D2-Clean	Synthetic-D2-Clean&Aug	Synthetic-D3-Clean&Aug
SIS ±Std	1.425±0.135	1.387±0.098	1.244±0.104	1.381±0.091	1.381±0.132

Generalization Ability

From Table 11.6, GA values of the pixel level are generally better than those of the object and the structural levels in most synthetic cases because of the more uniform, relatively simpler and less noisy patterns in the pixel level images. Consistent with human evaluation, images of the object and the structural levels are less clear and many undamaged state images may be mistaken as damaged, which confuses the classifier in damage detection. The exception is in the synthetic-D2-Clean case, where the GA of the structural level is only 0.03% higher than that of the pixel level. However, it does not significantly contradict the above-mentioned observations. Similar to scenario 1, after cleaning, the accuracy declines in the Synthetic-D2-Clean compared to the Synthetic-D2-All due to information loss from removing some data. On the contrary, DA and deeper representation slightly resolve this issue. However, poor results are observed in the structural level of the Synthetic-D3-Clean&Aug, with 20% reduction of the GA compared to the Synthetic-D2-Clean&Aug, indicating the instability of using a deeper representation for the damage detection scenario, which also springs from greatly reducing the data variety via cleaning and by selecting from more sub-classes of depth-3. Comparing with GA from Real-All, synthetic pixel level images show less than 5% difference under the best case of Synthetic-D2-Clean&Aug, indicating the small distribution difference towards the real dataset which is indicative of the effective performance enhancement of DL models using GANs. As for the object and the structural levels, there exist some discrepancies between the synthetic dataset and the real one. Thus, further improvements are needed through future research endeavors.

Table 11.6: GA for synthetic images in the damage state check

Dataset	**GA**		
	PD versus PUD	OD versus OUD	SD versus SUD
Real-All	94.76%	93.85%	97.5%
Synthetic-D2-All	87.90%	77.07%	81.82%
Synthetic-D2-Clean	78.83%	70.92%	78.86%
Synthetic-D2-Clean&Aug	89.31%	75.65%	86.14%
Synthetic-D3-Clean&Aug	88.10%	75.89%	66.14%

More discussions

In general, synthetic images generated from four cases indicate the promising results of using DCGAN with the LB method, which is beneficial to increasing the variety of dataset. As for visual quality from human judgement, using depth-2 representation with all raw images can achieve a stable baseline quality. Applying augmentation and using deeper representation lead to improvement with clear and sharp structural patterns. Moreover, as complementary metrics of human evaluation, results of SIS and GA are shown to be reasonable for GAN evaluation. IS has many criticisms and drawbacks [6]. However, due to the high accuracy of the pre-trained classifier with limited label categories, SIS indicates a stable performance consistent with the observation that an image with better visual quality achieves a higher score. As for GA, the trend indicates that with deeper representation and more data cleaning operations, better visual qualities are obtained but some information are lost leading to possible decrease of GA, which is a trade-off between quality and generalization. Using deeper representation leads to training more GAN models for new split leaf nodes, which introduces a large computational cost. Thus, how deep one needs to go depends on the complexity of the detection tasks at hand, e.g., depth-2 and depth-3 representations are sufficient for the two scenarios presented in this study.

As indicated in many previous studies, explaining the physics behind the outcome is one of the hardest problems in the current GAN research area. In this study, damage patterns in synthetic images are generated from some learned distributions, and it is possible that they may violate the laws of physics. However, comparing to previous studies, physical meanings have been assigned, to some extent, in the training procedure which somewhat relieves the issue via adding domain knowledge to the training pipeline, i.e., the ϕ-Net framework. In this manner, random structural images are filtered into more fine sub-categories and this guides the GAN to generate images representing specific type of structural images, which helps to reduce the complexity of the problem and to control the variance of the image quality. From the results presented for the pixel level and the damaged state of the Synthetic-D2-Clean case in Fig. 11.5, the GAN models learned to generate synthetic images not randomly; instead, these images are comparable to a combination of several basic crack features, i.e., diagonal cracks with different angles, widths, locations, and even interaction with spalling, which are judged to be realistic compared to the real damaged concrete structures. Moreover, the scope of using synthetic images similar to the conventional DA, discussed in this study, is only for the purpose of classification. However, finding the relationship between the synthetic damage images and the laws of physics is marked as future extensions. In addition, previous work has shown the potential of the GANs to solve this issue. From Radford et al. [97], by manipulating the initial value of the code z in the generator (Fig. 10.2), the content of the synthetic image can be controlled. Therefore, this observation shows the potential for future studies where a well-learned relationship of the code z (refer to Section 10.1) based on the laws of physics can help the GAN model to have a better understanding to generate images without violating these laws of physics. Accordingly, the image content can be controlled with the desired crack morphology, i.e., angle, damage level, spalling pattern, etc.

11.3 Experiments on Synthetic Data Augmentation for Classification

Experimental objectives

In the previous section, GAN was firstly validated and was shown to be able to generate reasonable structural images under certain settings. In this section, a second question arises, namely whether these synthetic images can alleviate the data deficit problem while using the DL approach, e.g., in classification tasks?

One straightforward application is to aggregate synthetic images to the real dataset to enhance the classifier performance in classification problems. However, preliminary investigation demonstrated that such aggregation may render worse performance due to three issues: (1) relatively lower quality of the synthetic images compared to the real ones, (2) possible distribution difference between the dataset of the generated images and the real dataset, and (3) feeding mixed data including synthetic ones into the classifier is thought to form a series pipeline by the GAN model and the classifier. This series pipeline introduces additional parameters compared to the original classifier, which may exacerbate the over-fitting issues. Thus, special training procedures and algorithms should be developed and investigated. This includes how to use synthetic data to improve the classifier performance under certain conditions. This needs to be explored as conducted in the following experiments.

According to the results presented in the previous sections, the validation accuracy of Real-All trained by classical DL architectures already achieved very high values of about 95% to 97%, where further improvement will not be so evident nor practical. Because in such level of accuracy, sometimes the so-called improvement is just one kind of over-fitting to the current dataset. Moreover, some recent studies validated the effectiveness of using synthetic images in a low-data regimes [4, 32, 102, 144]. Thus, in this section, a special case under the condition of a complex detection problem, low-data regime, and restricted computational power was designed and discussed. This special case was achieved by satisfying the following conditions:

1. Instead of conducting simple detection tasks (scene level and damage state classification) independently, considering both label information formed a six-class classification problem (PUD, PD, OUD, OD, SUD, and SD), which is thought of as a more complex problem.

2. D2-Clean dataset was treated as a training dataset with a decreasing number of images in training from 5,457 to 768 to increase the difficulty of the task, but the validation dataset remained the same as that used for GA computation.

3. A weaker classifier was designed and treated as a baseline model, where the term "weaker" was represented by using a few layers and a few parameters in the CNN to simulate the compromise made for limited computational resources.

Moreover, such complex six-class classification problem is meaningful as a practical application, which represents a complex environment and more autonomous setting.

Synthetic data fine-tuning (SDF)

Inspired by previous studies [32, 94], instead of directly applying more complex semi-supervised learning [102, 145] techniques, a fine-tuning pipeline, namely synthetic data fine-tuning (SDF), was proposed analogous to the previously discussed TL. In the SDF, a weak classifier is firstly pre-trained on the generated synthetic images and then fine-tuned by real ones, refer to Fig. 11.7.

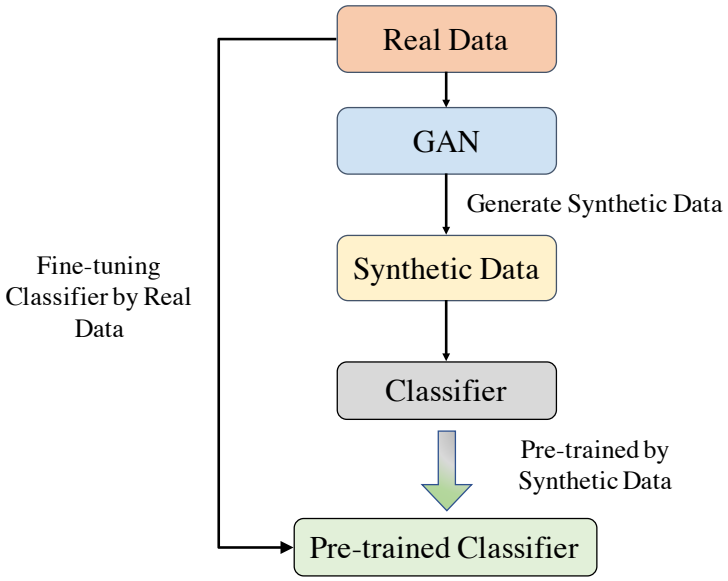


Figure 11.7: Illustration of the synthetic data fine-tuning (SDF) pipeline

Using a weak or non-classic CNN classifier restricted by the available computational power usually means that such classifier does not have pre-trained parameters from ImageNet or other source datasets in the source domain. Instead, its initialization fully depends on random, Gaussian, or other initialization approaches, e.g., Xavier initialization [36], which may be poor due to a large random parameter space. Conceptually, it is difficult to learn well the decision boundary through directly training from such initialization as illustrated in Fig. 11.8. When considering the SDF pipeline, regardless of the correctness of the generated synthetic images, these images can be thought of as being generated from a similar or enlarged sample space, which origins from raw real data. Therefore, it is believed that the classifier when pre-trained from such enlarged space can provide a better initialization for a fine-tuning procedure with the real data.

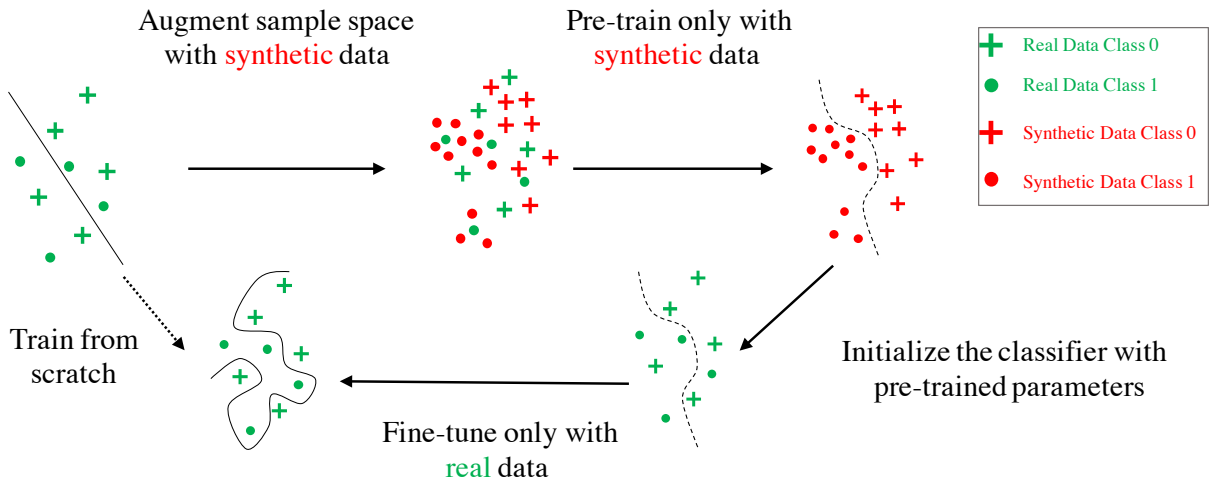


Figure 11.8: Two learning modes of conventional training and SDF

Experimental setup

In order to comprehensively explore the questions proposed in the experimental objectives, the comparative experiments were conducted for the following cases:

1. Train on D2-Clean.
2. Train on D2-Clean but using DA in the classifier training.
3. Train on D2-Clean aggregated with the Synthetic D2-Clean.
4. Same as Case 3 but using DA in the classifier training.
5. Train on D2-Clean aggregated with the Synthetic D2-Clean&Aug.
6. Same as Case 5 but using DA in the classifier training.
7. Pre-train on Synthetic D2-Clean and fine-tune on D2-Clean.
8. Pre-train on Synthetic D2-Clean and fine-tune on D2-Clean but using DA in the classifier training.
9. Pre-train on Synthetic D2-Clean&Aug and fine-tune on D2-Clean.
10. Pre-train on Synthetic D2-Clean&Aug and fine-tune on D2-Clean but using DA in the classifier training.

Cases 1 and 2 provide the baseline performance of training on small-scale dataset with and without using the DA technique. Cases 3 to 6 represent directly aggregating synthetic data

into real dataset, and cases 7 to 10 represent the SDF pipeline proposed above. Through these ten cases, the question about whether direct synthetic data aggregation and SDF pipeline can enhance the classifier's performance was explored. It is noted that in cases 3 to 6, for simplicity, 1:1 real to synthetic ratio was used. In other words, the same amount of synthetic data was randomly generated and mixed into the real dataset. Without loss of generality, especially towards cases 3 to 6, each case had 10 runs.

The weak classifier designed in this experiment was a shallow CNN constructed by only 2 Conv layers with 16 3×3 filters followed by one max pooling layer with stride 2, and then one fc-layer with 128 neurons activated by ReLU and one softmax layer whose output equals to the number of classes, i.e. 6 in this experiment. For a fair comparison, Adam optimization with learning rate 10^{-4}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ was applied for all cases. In the SDF pipeline, pre-training procedure took very few epochs and stopped when the validation accuracy no longer increased or continued to decrease in the following steps. Usually, the fine-tuning procedure took around 50 epochs until the results converged.

Experimental results & discussions

The experimental results are shown in Table 11.7, where some important observations are obtained. First, two baseline results in cases 1 and 2 show that the conventional DA improved the accuracy by only 4%. From the comparison between cases 1 to 6, it is obvious that direct synthetic data aggregation rendered a worse performance than the baseline, and DA operation led such aggregation to an unstable direction, where variance (Std) became larger in cases 4 to 6. It should be pointed out that in cases 4 to 6, the large variance does not mean that both extremely high and low values of the validation accuracy were achieved among multiple runs. The fact is that most results from 10 runs were close to the mean, but only 2 or 3 sets of results achieved relatively low accuracy, e.g., 34% in case 5. This can be partially explained by the fact that the conventional DA operation over-emphasized the lower quality synthetic data and diluted the contribution of the real data. Thus, simply mixing synthetic data into the real dataset did not work in this experiment. However, the idea of synthetic aggregation may work under some special training pipelines and algorithms, as reported in other cases [4, 102, 144]. From cases 7 to 10, the proposed method can obviously improve the performance, i.e., with different approaches of the pre-training, the proposed method shows accuracy improvement (54.2% to 56.4%) compared to that using only training on D2-Clean without any augmentation (49.6%, i.e. 4.6% to 6.8% enhancements) and compared to only 3.9% enhancement by the conventional DA (53.5%). Here it is reiterated that the parameters of the classifier in cases 1 to 6 were initialized from a random or Gaussian space, fine-tuning pipeline by pre-training the model based on the synthetic images achieved up to 6.8% accuracy improvement, which is viewed as remarkable in this experiment.

Based on this experiment, the appropriate usage of synthetic images can improve the classification performance of a weak classifier under the low-data regime and limited computational resources. While adopting the SDF pipeline, the weak classifier was firstly pre-trained on the synthetic images for the purpose of better initialization of the second fine-tuning

Table 11.7: Validation accuracy for the experiment using synthetic data aggregation cases

Case	Description	Accuracy	Run-time
1	Train on D2-Clean	49.6% ±0.2%	≈ 6 mins
2	Train on D2-Clean with DA	53.5% ±0.3%	≈ 9 mins
3	Train on D2-Clean + Synthetic D2-Clean	48.9% ±0.6%	≈ 7 mins
4	Train on D2-Clean + Synthetic D2-Clean with DA	49.6% ±3.4%	≈ 17 mins
5	Train on D2-Clean + Synthetic D2-Clean&Aug	47.1% ±6.5%	≈ 7 mins
6	Train on D2-Clean + Synthetic D2-Clean&Aug with DA	42.3% ±6.6%	≈ 17 mins
7	Pre-train on Synthetic D2-Clean and fine-tune on D2-Clean	55.2% ±0.2%	≈ 10 mins
8	Pre-train on Synthetic D2-Clean and fine-tune on D2-Clean with DA	54.2% ±0.3%	≈ 15 mins
9	Pre-train on Synthetic D2-Clean&Aug and fine-tune on D2-Clean	54.9% ±0.1%	≈ 10 mins
10	Pre-train on Synthetic D2-Clean&Aug and fine-tune on D2-Clean with DA	56.4% ±0.1%	≈ 15 mins

process, Fig. 11.7. It is even more effective than the conventional DA, and thought to be useful and applicable in many practical scenarios. However, as a trade-off, comparing to the conventional DA, the SDF pipeline is inefficient in two ways: (1) it is more time-consuming, and (2) it leads to insufficient usage of the generated data. From Table 11.7, regardless of the hours of computational costs in the GAN training[3], the computational cost in the SDF pipeline without (cases 7 & 9) and with (cases 8 & 10) adoption of the DA in training the weak classifier are nearly double and triple of that in the baseline (case 1), respectively. However, the conventional DA (case 2) only takes 1.5 times of that in the baseline. On the other hand, in the SDF pipeline, over thousands of synthetic data were generated, but they were only utilized in the pre-training process once, and then discarded. Considering the high-cost in training the GAN model, and generating and storing the synthetic images, such low utilization rate is thought of as a waste of resource. Therefore, considering recent works documented in [94, 144], more efficient GAN and training pipeline should be studied in the future.

11.4 Summary

In this chapter, two types of GAN-based experiments were conducted, namely synthetic structural image generation and synthetic data augmentation in classification, where two scenarios (scene level identification and damage state check) were considered. The following key conclusions are inferred from all the conducted experiments:

- DCGAN is a stable GAN model which can generate reasonable structural images (by human evaluation) with the LB method applied for all levels even when only using a depth-2 representation framework without data cleaning, and it outperforms the directly training schema of GAN.

[3]This is because once GAN is trained, the generating/sampling process for data augmentation is fast and can be utilized repeatedly without retraining the GAN.

- Using cleaned dataset produces synthetic images with clearer objects but increases difficulties in the training procedure and mode collapse issues are more likely to occur.
- Conventional DA and deeper representation improve the visual quality in most cases.
- Consistent with human observation, better image quality leads to a higher SIS value. On the other hand, GA shows a trade-off between image quality and generalization.
- The SDF pipeline using synthetic data can enhance the performance of a weak classifier under the low-data regime and limited computational resources.

The following items are suggested as future extensions:

– There are many types of GAN models, e.g., Wasserstein GAN (W-GAN) [5], Conditional GAN [81], etc. Therefore, selection of the GAN model as a hyper-parameter is worth exploring. Moreover, the methods and results presented in this dissertation can be considered as benchmarks and references for future comparative works.

– There are many metrics to be explored, e.g., Wasserstein Distance for applying WGAN and Fréchet Inception Distance (FID) proposed by [44].

– While adopting synthetic data aggregation for classification problems, SDF pipeline is time-consuming and inefficient with generated data compared to conventional DA, and more explorations are expected by using the techniques of *semi-supervised learning* [102, 118]. This is explored in Chapter 12.

– More data aggregation applications, other than classification, need to be explored, e.g., damage localization, semantic segmentation, etc.

– The major drawback of applying the current GANs into engineering applications, e.g., damage classification, is that the generated synthetic data fully depend on the optimization by minimizing the distribution difference with the real data, and it may not fully comply with the laws of physics of the problem at hand. Thus, future studies should consider to encode physics principles and information into the GAN model formulation.

Chapter 12

Balanced Semi-Supervised GAN (BSS-GAN) for Structural Image Classification

From the experiments in Chapter 11, the effectiveness of GAN in generating reasonable structural images under certain training strategies and settings was validated. Moreover, to overcome the limitations of directly mixing synthetic images with real ones for training, a special union training pipeline was proposed, namely synthetic data fine-tuning (SDF). Such training pipeline was able to enhance the classifier performance by nearly 7% over the baseline. However, as concluded, this training pipeline is still inefficient, because both the GAN model and the classifier need to be trained, and large amounts of synthesized images are left valueless after the pre-training process. According to the study of [75, 76], reformulating the problem into a *semi-supervised learning* paradigm seems to be a promising solution.

In this chapter, the GAN is converted into a semi-supervised variant, where the semi-supervised learning mechanism can more thoroughly exploit the features of the unlabeled data, and simultaneously increasing the model's synthetic data generation and classification capabilities. Furthermore, besides considering the restrictions of the low-data regime and limited computational power, imbalanced-label/class issues are explored through experiments. To better address the imbalanced-class issues, a balanced batch sampling technique is adopted in the training procedure, and the whole model is named as Balanced Semi-supervised GAN (BSS-GAN).

12.1 Formulation of Balanced Semi-supervised GAN (BSS-GAN)

The original GAN is trained in an unsupervised learning manner, and its discriminator only differentiates unlabeled real samples from those synthesized by the generator. Herein, the output dimension (size) of the discriminator increases from 2 ("real" or "synthetic") to $K+1$

such that the discriminator can classify samples from K real classes (for samples in P_{Data}) and one "generated" class (for samples in P_G). The model can now learn from both unlabeled and labeled data.

The discriminator D takes in a sample x and outputs a $K+1$-dimensional vector of class probabilities $p_{model}(y|x)$, where $p_{model}(y=i|x) = \exp(D(x)_i)/\sum_{j=1}^{K+1}\exp(D(x)_j)$ represents the probability of the i-th class and $D(x)_1, \ldots, D(x)_{K+1}$ are logits corresponding to each class. Herein, $p_{model}(y=K+1|x)$ represents the predicted probability that sample x is "synthetic", and $D(x)$ in Eq. 10.1 can be substituted by $1-p_{model}(y=K+1|x)$. Similarly, $1-D(G(z))$ in the second term of Eq. 10.1 is equivalent to $p_{model}(y=K+1|x)$. In order to minimize the discriminator loss, the value function $V(\cdot)$ in Eq. 10.1 is reformulated into Eq. 12.1 with negative signs and the above substitutions. Now, the computation becomes unsupervised without using label information.

$$L^{(D)}_{unsupervised} = -E_{x\sim P_{Data}}\log(1-p_{model}(y=K+1|x)) - E_{x\sim P_G}\log(p_{model}(y=K+1|x)) \quad (12.1)$$

The supervised loss of the discriminator is the cross-entropy between K real class labels (i.e., $y=i, i<K+1$) and the model's predicted class distribution $p_{model}(y|x)$:

$$L^{(D)}_{supervised} = -E_{x,y\sim P_{Data}(x,y)}\log(p_{model}(y=i|x, i<K+1)) \quad (12.2)$$

Finally, the total discriminator loss is expressed as follows:

$$L^{(D)} = L^{(D)}_{unsupervised} + L^{(D)}_{supervised} \quad (12.3)$$

The generator's gradient vanishes when the discriminator fully distinguishes the synthetic samples from the real ones, i.e., when $D(G(z)) \to 0$. Goodfellow [37] proposed a heuristically motivated GAN game loss for the generator as follows:

$$L^{(G)} = -\frac{1}{2}E_{z\sim P_Z}\log(D(G(z))) \quad (12.4)$$

where $E_{z\sim P_Z}$ is as expressed in Eq. 10.1.

Instead of minimizing the expected log-probability of the discriminator being correct, the generator now maximizes the expected log-probability of the discriminator making a mistake, i.e., labeling a generated sample $G(z)$ as real. In this study, the constant multiplier, $1/2$, is dropped and the expression is modified to fit the semi-supervised scenario with $K+1$ classes as follows:

$$L^{(G)}_{heuristic} = -E_{x\sim P_G} log(1-p_{model}(y=K+1|x)) \quad (12.5)$$

From Salimans et al. [102], feature matching is a technique that prevents over-training the generator and increases the stability of the GANs. The generator is trained to output samples which yield similar activation at an intermediate layer of the discriminator as those from the real samples as follows:

$$L^{(G)}_{feature\ matching} = \|E_{x\sim P_{Data}} f(x) - E_{z\sim P_Z} f(G(z))\|_2^2 \tag{12.6}$$

where $f(x)$ is the activation of an intermediate layer of the discriminator for a given sample x. In this study, the flattened output (activated by ReLU) of the last Conv layer in the discriminator network is chosen as $f(x)$. Finally, the total generator loss is computed as follows:

$$L^{(G)} = L^{(G)}_{heuristic} + L^{(G)}_{feature\ matching} \tag{12.7}$$

In order to address the strong imbalanced-class issue, a special training procedure using the idea of *balanced batch sampling* is proposed. Instead of randomly sampling data from the raw dataset to form the training batch, where the class ratio is extremely biased, samples from each class are equally drawn to form the batch with a balanced class ratio. Such strategy may be helpful in eliminating the prior bias during training. This whole pipeline including Semi-supervised GAN and balanced batch sampling is named the Balanced Semi-supervised GAN (BSS-GAN). For a training batch size of $2N$, the detailed procedure is listed below:

Step 0: Initialize the discriminator D and the generator G.

Step 1: Half the batch of real samples, $x_1, \ldots, x_N$, is passed into the discriminator, and these samples are equally drawn from K classes according to the labels. For each $x_i \in \{x_1, \ldots, x_N\}$, the discriminator outputs a K+1-dimensional probability vector, $v_i = [p_{model}(y = j|x_i), \forall j \in \{1, \ldots, K+1\}]^T$.

Step 2: Half the batch of random noise vectors, $\{z_1, \ldots, z_N\}$, is fed into the generator. For each $z_i \in \{z_1, \ldots, z_N\}$, the generator outputs a generated sample, $G(z_i)$.

Step 3: The generated synthetic samples, $\{G(z_1), \ldots, G(z_N)\}$, are passed into the discriminator. For each $G(z_i) \in \{G(z_1), \ldots, G(z_N)\}$, the discriminator outputs a K+1-dimensional probability vector $u_i = [p_{model}(y = j|G(z_i)), \forall j \in \{1, \ldots, K+1\}]^T$.

Step 4: Optimize and update the parameters of the discriminator and the generator, θ_D and θ_G, respectively, by minimizing Eqs 10.7 & 10.8.

Repeat steps (1) to (4) until convergence is achieved or the designated number of epochs is reached.

12.2 Experimental Objectives

In this study, one basic ϕ-Net benchmark task, namely Task 3 for the spalling condition detection, was selected for validation experiments, because the class label ratio between "spalling" (SP) and "non-spalling" (NSP) is usually imbalanced in practice. Two major topics were studied herein, namely (1) the effects of different label ratios in the training set, and (2) the effects of different training pipelines.

For topic (1), different label ratios represent different amounts of labeled data that can be used for training, since conventional CNN can only utilize labeled data in supervised learning. For topic (2), the difference between the three types of training pipelines: (i) conventional training, (ii) conventional over-sampling by augmenting the minority class with GAN-generated data, and (iii) training in the format of semi-supervised learning, is investigated.

12.3 Experimental Setup

Dataset

In this study, the subset of Task 3 spalling condition in the ϕ-Net was used as the raw data source, and it was further processed for the designed experiments. Data preprocessing was conducted as follows:

1. Clean the dataset and select pixel level images with high visual quality.
2. Select and store the NSP and SP images in a roughly 4:1 ratio to build the experimental dataset.
3. Resize the images to 128×128 pixels.
4. Split the data into training and test datasets with a 2:1 ratio.

Finally, a dataset with a total of 2,084 images was constructed, and the label statistics are shown in Table 12.1. Compared with the ϕ-Net benchmark experiments and general CV applications, the dataset used herein is considered imbalanced and under the low-data regime.

Table 12.1: Label statistics of the dataset for the spalling condition classification

Label	**Training set**	**Test set**	**Split ratio**
NSP	1,130	565	2
SP	259	130	2
Class ratio	4.4	4.3	-

Network configuration

For the spalling condition classification, a conventional multi-layer CNN discriminative classifier, denoted as the baseline (BSL), was designed and presented in Table 12.2. For a fair comparison, the discriminator of the BSS-GAN used the same architecture as the BSL, e.g., number of layers and filters, filter sizes, etc., except for the last softmax layer. The configurations are summarized in Table 12.2 for the discriminator of the BSS-GAN

and also for the BSL. According to findings in Radford et al. [97], Leaky ReLU was used as the activation function with negative slope coefficient $\alpha = 0.2$, and batch normalization (BatchNorm) layers were inserted after the intermediate Conv layers with momentum 0.8. To avoid over-fitting, dropout was also applied with a 0.25 dropout rate.

The configurations of the generator of the BSS-GAN are shown in Table 12.3. Because the real image size is only 128×128, a conventional 100-dimensional noise vector was generated from Normal distribution as the input to the generator. There is no dropout in the generator, but BatchNorm layers with momentum 0.8 were added after the deconvolutional (Deconv) layers except for the last one.

Table 12.2: Configurations of the discriminator of the BSS-GAN and also of the BSL

Layer	**Filter size (#)**	**Activation**	**Shape**	**Notes (α: −ive slope coef. in Leaky ReLU)**
Input	-	-	(N, 128, 128, 3)	Input RGB images of size 128×128
Conv	3×3 (32)	Leaky ReLU	(N, 64, 64, 32)	Stride = 2, $\alpha = 0.2$
Dropout	-	-	(N, 64, 64, 32)	Dropout rate = 0.25
Conv	3×3 (64)	Leaky ReLU	(N, 32, 32, 64)	Stride = 2, $\alpha = 0.2$
BatchNorm	-	-	(N, 32, 32, 64)	Momentum = 0.8
Dropout	-	-	(N, 32, 32, 64)	Dropout rate = 0.25
Conv	3×3 (64)	Leaky ReLU	(N, 32, 32, 64)	Stride = 1, $\alpha = 0.2$
Flatten	-	-	(N, 65,536)	65,536 = 32×32×64
Fc-layer	-	Softmax	(N, K)	BSS-GAN: $K = 2 + 1$; BSL: $K = 2$

Table 12.3: Configuration of the generator of the BSS-GAN

Layer	**Filter size (#)**	**Activation**	**Shape**	**Notes**
Input	-	-	(N, 100)	Noise generated from Normal distribution
Fc-layer	-	ReLU	(N, 131,072)	131,072 = 32×32×128
Reshape	-	-	(N, 32, 32, 128)	-
Deconv	3×3 (128)	ReLU	(N, 64, 64, 64)	Stride = 2
BatchNorm	-	-	(N, 64, 64, 64)	Momentum = 0.8
Deconv	3×3 (64)	ReLU	(N, 128, 128, 3)	Stride = 2
BatchNorm	-	-	(N, 128, 128, 3)	Momentum = 0.8
Deconv	3×3 (3)	ReLU	(N, 128, 128, 3)	Stride = 1

Training configuration

To investigate topic (1), four magnitudes, namely 10%, 25%, 50%, and 100% of the training data, were used for supervised training in both the BSL and the BSS-GAN. It is noted that the labeled images were selected randomly, but the class label ratio (4:1) was still maintained. Moreover, for BSS-GAN training, the remaining data were treated as unlabeled and also fed

into the network for semi-supervised training along with the labeled data. To control the experimental settings, both models were trained for 500 epochs with a batch size of 64. The optimizer was Adam [53] with an initial learning rate of 2×10^{-5}. The evaluation metrics were the overall accuracy, Eq. 2.7, and the true positive rate (TPR), Eq. 2.9, where SP and NSP were defined as "Positive" and "Negative", respectively.

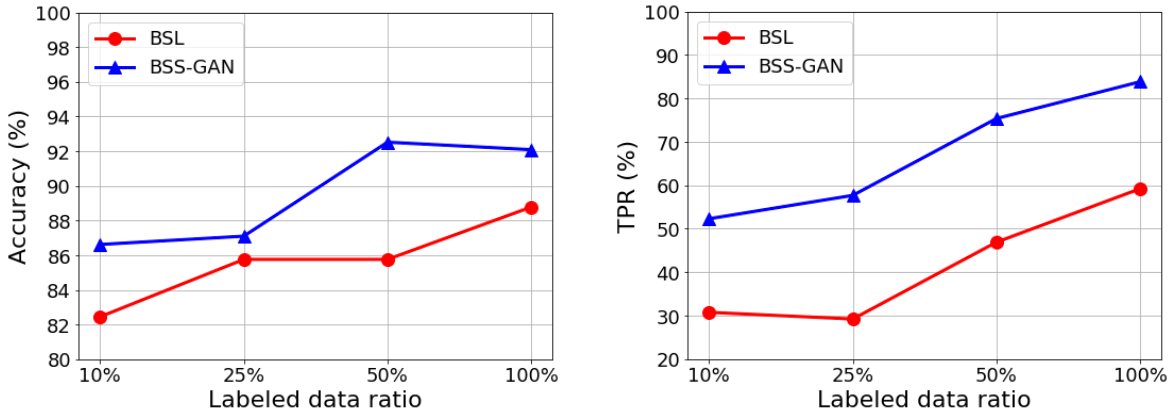
To investigate topic (2), an ordinary DCGAN model was trained for generating synthetic SP images, which shares the same network architecture with BSS-GAN, except for the last softmax layer and the loss function. Specifically, the generator loss in this ordinary DCGAN model was Eq. 10.5, while the generator loss of the BSS-GAN was Eq. 12.7. In addition, only the SP training images were fed into the ordinary DCGAN model. The total training epochs was 80,000 by Adam optimization with the same learning rate of 2×10^{-5}. A total of 871 synthetic SP images were manually selected from random sampling to augment the raw SP images. Finally, the dataset was over-sampled as a balanced dataset, where the number of training data for both classes was 1,130. In this experiment, comparisons were made between BSL, BSL with GAN-based over-sampled balanced dataset (denoted as GAN-Aug), and BSS-GAN with 100% labeled training data.

The experiments in this chapter were conducted on the TensorFlow platform. They were performed on CyberpowerPC with single GPU (CPU: Intel Core i7-8700K@3.7GHz 6 Core, RAM:32GB & GPU: Nvidia Geforce RTX 2080Ti).

12.4 Experimental Results & Discussions

Effect of the amount of labeled data

As the main purpose of this experimental study, the classification performance of the BSS-GAN was investigated and compared with BSL under varying amounts of labeled training data, and the results are shown in Table 12.4 and Fig. 12.1. This figure shows that as the number of labeled data increased, both models had increasing trends of accuracy and TPR, but the BSS-GAN outperformed the BSL for all cases. Due to the strong class imbalance in the experiment dataset, simply guessing that all data are NSP leads to 81.29% overall accuracy. When only using low amounts (10% & 25%) of labeled data, even though the test accuracy values of both models were slightly over 81.29%, the results were not satisfactory. From Table 12.4, it is observed that the TPR of the BSS-GAN was slightly over 50% and the BSL was much worse (only detected 30% of the SP images). This indicates that both models have strong biases to the majority NSP class. At 50% labeled ratio, the BSS-GAN had an improved performance with 6.76% increase over the BSL in the overall accuracy and 75.38% in TPR. At 100% labeled ratio, the overall test accuracy of the BSS-GAN maintained the same level at around 92% (ignoring the 0.43% drop due to numerical randomness), but the TPR of the BSS-GAN reached 83.85%, which is significantly higher than the BSL. In other words, while feeding more labeled data to the BSS-GAN, it became more sensitive and robust to detect the minority SP class with the trade-off of a slight accuracy decrease in NSP,



(a) Overall test accuracy (b) TPR

Figure 12.1: Trends of BSS-GAN vs BSL for increasing amounts of labeled data

i.e., the model became more conservative by treating more NSP as SP. Such performance enhancement is meaningful and beneficial to SHM, since high TPR and overall accuracy indicate a safer and more conservative model in practice.

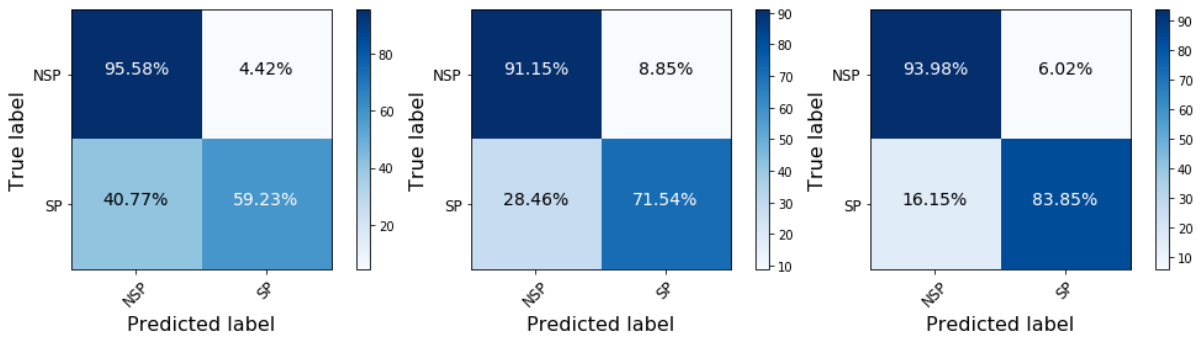
Table 12.4: Results (%) of BSS-GAN and BSL for varying amounts of labeled data

Metric	**Model**	**Ratio of labeled data**			
		10%	25%	50%	100%
Overall Test Accuracy	BSL	82.44	85.76	85.76	88.78
	BSS-GAN	86.62	87.10	92.52	92.09
TPR	BSL	30.77	29.34	46.92	59.23
	BSS-GAN	52.31	57.69	75.38	83.85

Even though both BSL and BSS-GAN share similar discriminative nature, the dynamic data interaction hidden in the BSS-GAN, and the contribution from the unlabeled images for feature learning significantly enhanced the performance of the BSS-GAN's discriminator. Specially, the dynamic data interaction means feeding the discriminator with new and different synthetic data produced by the generator in each training iteration. This can be viewed as one type of realization of the general DA method. In conclusion, the results validated the effectiveness of using GAN with a semi-supervised learning mechanism under the conditions of data deficiency and strong class imbalance, which alleviates the bias issues experienced by the conventional CNN classifiers.

Effect of the training pipeline

Based on the conducted experiments with the GAN-augmented dataset, the overall accuracy of GAN-Aug was 87.49%, and the TPR was 71.54%. Furthermore, plots of the confusion

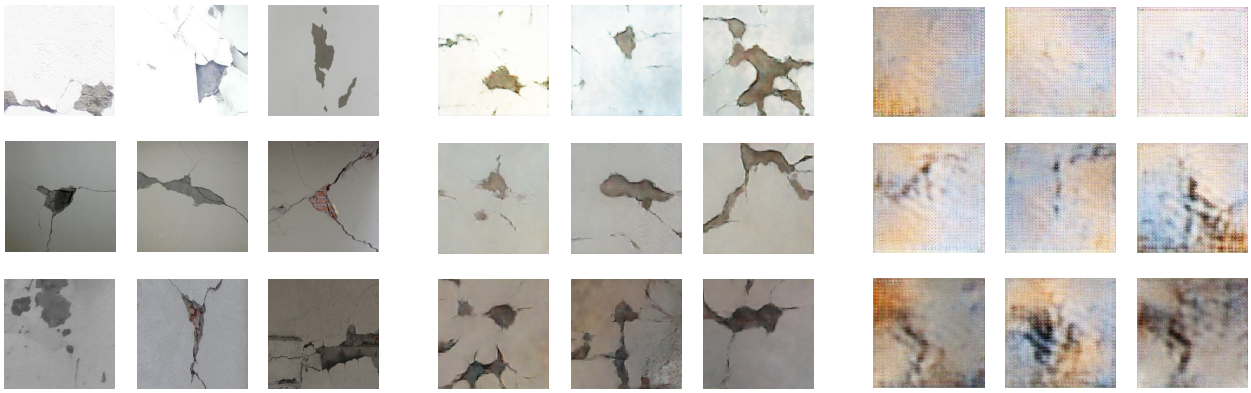


(a) BSL (b) GAN-Aug (c) BSS-GAN

Figure 12.2: Confusion matrices of different pipelines

matrices with respect to three pipelines are presented in Fig. 12.2. Comparing to the BSL and the BSS-GAN models, the overall test accuracy of the GAN-Aug was lower than both of them and the TPR was between the two. The observation that the GAN-Aug outperformed the BSL can be partially explained through using a balanced dataset where the model becomes less likely to be trained biased towards a single class. On the contrary, the BSL was trained on a strongly biased training set, which led to a biased model by simply predicting all data to be in the majority class. Similar to the BSS-GAN, the increased accuracy in detecting SP (e.g. as shown in the TPR) made the GAN-Aug model less sensitive to the NSP images. However, the accuracy improvement in the SP did not compensate of the accuracy decrease in the NSP, which caused a decrease in the overall accuracy. In other words, even though the GAN-Aug classifier achieved a better TPR than BSL, it lost more capability to recognize the NSP images, which did not make it a better model than BSL. Therefore, this experiment provided another example showing that simply over-sampling the raw dataset with GAN-generated data may not work. On the contrary, in this experiment, BSS-GAN achieved both higher overall test accuracy and TPR than the other two models, making it the best within the scope of this limited experiment GAN-related pipeline.

For computational efficiency, the training time costs for BSL, BSS-GAN, and GAN-Aug with the full dataset were approximately 0.75 hour, 1.5 hours, and 2 hours, respectively. In general, BSL consumed the least time, but considering its poor performance, this efficient computational time is not very informative or decisive. Compared to BSL, BSS-GAN needed to train extra parameters in the generator, and the generator also needed additional computational resources to synthesize images from the noise vectors. These additional operations demanded extra time costs, but they were still acceptable. A useful and informative comparison is that between BSS-GAN and GAN-Aug, since both are GAN-based augmentation methods. When compared with the GAN-Aug, the BSS-GAN consumed 25% less time, which indicates the computational efficiency of the BSS-GAN over the other GAN-based methods.



(a) Real for SP (b) Synthetic from GAN-Aug (c) Synthetic from BSS-GAN

Figure 12.3: Sample images from the real dataset and the synthetic dataset using the GAN-based pipelines

Comments on the synthetic image quality

In this section, synthetic images generated by both BSS-GAN and GAN-Aug are shown in Fig. 12.3. In general, no mode collapse issues (the generator produces limited varieties of images) occurred in either models and the obtained synthetic images had a good visual variety. However, the visual quality of images generated by GAN-Aug, Fig. 12.3b was much better than that of the BSS-GAN, Fig. 12.3c. This observation can be explained by several factors.

Unlike several mechanisms in ordinary GAN, the loss function in BSS-GAN focuses more on the classification than the image generation quality. This is reflected by the supervised cross-entropy loss. On the contrary, the ordinary GAN only includes the unsupervised loss, which is more about feature learning than classification. These different training objectives influence the performance of the generator even though the same generator architecture is used. Therefore, it can be inferred that the improvement in the discriminator's classification abilities may weaken the generator when the classification task of the discriminator increases from binary (GAN or GAN-Aug) to the $K + 1$ (BSS-GAN) classes.

As stated in Chapter 4, structural images have a complex and mixed distribution, which makes it difficult for ordinary GAN to generate clear and class-discriminative images. However, through conditioning, e.g., considering class information related to a specific class of images, makes the ordinary GAN able to generate higher quality images towards that class. In the GAN-Aug pipeline, only feeding pixel level SP images during the GAN training can be viewed as one conditioning operation, which significantly reduces the complexity of the data distribution. Thus, in Fig. 12.3b, the synthetic SP images show very realistic visual qualities. On the contrary, the generator in the BSS-GAN was trained with both NSP and SP images in an unsupervised manner. Thus, it had to learn a mixed distribution from both NSP and SP. As a result, the synthetic images generated by the BSS-GAN can be viewed as either in NSP, SP, or even in an intermediate (mixed) state. To show this, in Fig. 12.3c, synthetic

images in the first row are smooth and resemble NSP, and the remaining images in the second and third rows show some blurry and damage-like patterns, e.g., cracks and dark spots.

In summary, if only focusing on the synthetic image quality, the ordinary DCGAN with class conditioning has a better image generation ability than the BSS-GAN under the specified image resolution of 128×128 pixels. However, according to the experimental feedback from the classification performance, it can be inferred that even though synthetic images by the BSS-GAN are not clear enough and less realistic, the learned features and patterns from these generated images may still contribute to the learning process of the discriminator in the semi-supervised learning pipeline. This improves both the training efficiency and the discriminative performance, which are important enhancements for SHM decision making under the inevitable practical conditions of data deficiency and class-imbalance.

12.5 Summary

In this chapter, formulating GAN model into a semi-supervised framework was discussed. In order to improve the efficiency of using GAN in SHM classification tasks, a semi-supervised learning GAN pipeline (BSS-GAN) along with the balanced batch sampling technique was proposed. Under the imbalanced-class and low-data regime, a series of computer experiments with respect to reinforced concrete cover spalling detection, which is one of the most basic damage assessment tasks in vision-based SHM, were designed and conducted.

To investigate the discriminative capability, under different amounts of labeled data, the BSS-GAN was compared with the BSL, which shares the same discriminative network architecture. It was observed that the BSS-GAN outperformed the BSL in both overall accuracy and TPR at all labeled ratios (especially at 50%), which validated the effectiveness of the BSS-GAN under the above-mentioned constraints. Furthermore, using the whole training dataset as labeled data, the BSS-GAN was compared with another GAN-based pipeline, namely GAN-Aug. This third model over-sampled the minority class (SP) with synthetic data generated by an ordinary DCGAN until reaching the same amount of the majority class (NSP) and subsequently the BSL was trained with this balanced dataset. The results indicated that the GAN-Aug improved the TPR compared to the BSL by using more SP images, but it was still ∼10% lower than that of the BSS-GAN. Moreover, GAN-Aug sacrificed the performance in detecting NSP images, which led to the decrease of the overall accuracy, being far lower than that of the BSS-GAN. Therefore, in these limited experiments, BSS-GAN achieved the best classification performance over both BSL and GAN-Aug, with the exception of the quality of the generated images where BSS-GAN produced less realistic quality of the synthetic images than those from the GAN-Aug.

As a preliminary study, there still exist several factors that need to be explored. By computer experiments, BSS-GAN was only shown to be effective under the low-data regime, more experiments need to be conducted for medium-data regimes or even for large datasets, which can be realized by continuing the efforts in collecting and labeling structural images in practice, i.e. expanding the size of the ϕ-Net. Moreover, only one CNN architecture was

tested herein, and more parametric studies with respect to the network architecture are expected in the future.

In conclusion, the presented promising results by BSS-GAN shed light on the high potentials of Semi-supervised GAN in vision-based damage assessment and SHM, in general. This is clearly worth significant explorations and research efforts in the future to overcome two important practical hurdles faced by the SHM community, namely data deficiency and class-imbalance.

Chapter 13

Fusion of Time Series & Machine Learning

The previous chapters discussed several applications of the DL techniques in one data-driven SHM, namely vision-based SHM. This chapter and the next one explore a more general ML approach for another SHM direction, namely vibration-based SHM.

The data source of vibration-based SHM is in the format of *time series* (TS), which is defined by a series of data points indexed in ordered times [41]. Usually, TS is a sequence taken at successive equally spaced points in time, which is also a sequence of discrete-time data. In the domain of vibration-based SHM, vibration signals from deployed sensors, e.g., accelerometers to measure acceleration, velocimeters to measure velocity, and linear potentiometers, laser scanners, or even satellites to measure displacement, produce different realizations of TS. As stated in Chapter 1, TS using a family of autoregressive (AR) models was shown to be effective in damage detection and has been used to capture damage features from instrumented or simulated structures, which provides the foundation for the integration of TS and ML techniques.

In this chapter, the basic properties of TS, e.g., autocorrelation, stationarity, and definitions of the AR family models, are briefly introduced first. Then, to combine both TS modeling and ML classification together, an integrated end-to-end algorithm/framework, namely *autoregressive integrated moving average with machine learning* (ARIMA-ML), is proposed, which consists of four parts: (1) smoothing-segmentation-normalization-differencing (SSND), (2) auto-stationarity check, (3) feature extraction, and (4) ML classification modules. More details about these four parts are discussed in the following sections.

13.1 Basics of Time Series Modeling

In TS analysis, several properties are important, e.g., stationarity, autocorrelation, and partial autocorrelation. In this section, a brief review is given, and notations are defined as follows: X is one TS signal with duration T, and s and t are two particular time steps in this signal,

i.e. $s, t \in [0, T]$, with values expressed as X_s and X_t. Details can be found in [22, 34, 41, 111].

Autocorrelation

Autocorrelation, also known as serial correlation, is the correlation of a signal with a delayed copy of itself as a function of the delay [121]. Informally, it represents the similarity between observations X_s and X_t as a function of the time lag ($|s - t|$). The analysis of the autocorrelation is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies [121]. It is often used in signal processing for analyzing functions or series of values, such as time domain signals.

Before defining the autocorrelation function (ACF), the autocovariance function $\gamma(s, t)$ is introduced, which is also known as second moment product for all s and t with mean values μ_s and μ_t at these respective times. Note that E is the expectation operator in Eq. 13.1.

$$\gamma(s, t) = cov(X_s, X_t) = E[(X_s - \mu_s)(X_t - \mu_t)] \tag{13.1}$$

The autocovariance function measures the linear dependence between two points on the same series observed at different time steps. Then, the ACF, $\rho(s, t)$, with respect to observations X_s and X_t is defined as a normalized version of the autocovariance function $\gamma(s, t)$ as follows:

$$\rho(s, t) = \frac{\gamma(s, t)}{\sqrt{\gamma(s, s)\gamma(t, t)}} \tag{13.2}$$

Therefore, the ACF defines the relationship between two points in time with a measure of association ranging between -1 and 1.

Partial autocorrelation

The partial autocorrelation function (PACF) gives the partial correlation of a stationary TS with its own lagged values, regressing the values of the TS at all shorter lags [121]. It contrasts with the ACF, which does not control other lags. This function plays an important role in data analysis aimed at identifying the extent of the lag in an AR model.

Consider a TS, X, the PACF is defined as the correlation between X_t and X_{t+k} after removing the effect of the intervening variables $X_{t+1}, X_{t+2}, \ldots, X_{t+k-1}$. Consider predicting X_t as $\hat{X}_t$ based on a linear function of these intervening variables. Accordingly, the estimated term is expressed as $\hat{X}_t = \beta_1 X_{t+1} + \beta_2 X_{t+2} + \cdots + \beta_{k-1} X_{t+k-1}$, with β's chosen to minimize the mean square error (MSE) of the estimation by thinking forward in time. Assuming that the β's have been so chosen and then thinking backward in time, it follows from the stationarity (defined in the following subsection) that the best estimator of X_{t+k}, based on the same intervening variables, can be formulated as $\hat{X}_{t+k} = \beta_1 X_{t+k-1} + \beta_2 X_{t+k-2} + \cdots + \beta_{k-1} X_{t+1}$, refer to [111].

The PACF at lag k is defined as the correlation between the prediction errors, making use of the coefficients, $\phi(k)$, namely the partial autocorrelations at lag $k \geq 2$, as follows:

$$\begin{aligned}\phi(k) &= corr(X_{t+k} - \hat{X}_{t+k}, X_t - \hat{X}_t) \\ &= corr(X_{t+k} - \beta_1 X_{t+k-1} - \cdots - \beta_{k-1} X_{t+1},\ X_t - \beta_1 X_{t+1} - \cdots - \beta_{k-1} X_{t+k-1})\end{aligned} \tag{13.3}$$

Stationarity

A stationary TS is the one whose statistical properties, e.g., mean, variance, and autocorrelation are constants over time. Most statistical forecasting methods are based on the assumption that the TS can be rendered approximately stationary through certain mathematical transformations.

In statistics, the stationarity can be defined in two ways: (1) *strictly stationary* and (2) *weakly stationary*. Usually strictly stationary is too strong and hard to satisfy for most practical applications. Thus, only weakly stationary is introduced and used as one of the evaluation criteria in this dissertation. For simplicity, "weakly" is dropped and is simply referred to as "stationary" in the following text. According to [111], a weakly stationary TS, X, is a finite variance process such that:

1. The mean value function, μ, is constant and does not depend on the time step t.
2. The autocovariance function, $\gamma(s, t)$, depends on time steps s and t only through their differences or lag, i.e., $|s - t|$.

Many models and properties are required for a stationary TS. Thus, checking stationarity of the test TS is important. Instead of implementing the above criteria to check the stationarity manually, there are many other ways to do so numerically. The *Augmented Dickey–Fuller* (ADF) test is one of the widely used methods, especially in statistics and economics, and it is introduced in a later section of this chapter.

AR model family

In classical TS modeling, the relationships between different variables are established through regression analysis and links between different target observations. AR series modeling is a family of regression models used for general TS problems [111], which includes AR, moving average (MA), autoregressive moving average (ARMA), AR with exogenous (ARX) input, autoregressive integrated moving average (ARIMA), etc. It is noted that there are several SHM applications using these models but with fewer cases of ARIMA, which is the focus of the study presented in this dissertation.

AR(p) model

Denote the basic AR model with order p as AR(p), which aims to determine the dependent relationship of an observation with the order p of lagged observations. Suppose that X is a stationary TS, its value at time t, X_t, can be represented by certain combinations of the values at the previous p time steps and associated parameters, where α's are the parameters of the model to be determined by training ($\alpha_p \neq 0$), with white noise error ϵ_t and constant c, Eq. 13.4. Usually, ϵ_t is assumed as a Gaussian white noise with a zero mean and a finite variance.

$$X_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + \epsilon_t + c \tag{13.4}$$

MA(q) model

Similarly, denote the MA model with order q as MA(q), which aims to determine the dependent relationship of the observation with order q of observed white noise error terms. Thus, X_t can be represented by the mean of TS, μ, certain combinations of current and previous q noise terms, and parameters β's, Eq. 13.5.

$$X_t = \mu + \epsilon_t - \sum_{i=1}^{q} \beta_i \epsilon_{t-i} \tag{13.5}$$

It is noted that Eq. 13.5 is a general form. Through preprocessing like centralizing the data, i.e., $X'_t = X_t - \mu$, it can be simplified as follows:

$$X_t = \epsilon_t - \sum_{i=1}^{q} \beta_i \epsilon_{t-i} \tag{13.6}$$

ARMA(p, q) model

Extending from the formulations of the AR and the MA models, several combined models are proposed. The ARMA(p, q) model is thus a *linear combination* of AR(p) and MA(q) models, as follows:

$$X_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + \epsilon_t - \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + c \tag{13.7}$$

ARIMA(p, d, q) model

In many cases, it is found that a TS may achieve stationarity after several differencing operations. Thus, the AR series models, e.g., ARMA, may fit an arbitrary TS after d times of differencing. In other words, the raw TS can be estimated by d times integration of the

fitted TS. This explains the word "integrated" in the name of the ARIMA model. In the implementation of the ARIMA(p, d, q) model, the raw TS is subjected to d times differencing (differentiation) first, then fitted with an ARMA(p, q) model.

Suppose X satisfies the stationarity condition after the d times differencing of Y, i.e., $X = \nabla^d Y$. Therefore, Y follows an ARIMA(p, d, q) model, and X follows an ARMA(p, q) model, where for each time step t, X_t can be expressed as follows:

$$X_t = \nabla^d Y_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + \epsilon_t - \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + c \tag{13.8}$$

For more information about other variants of the AR model, refer to [22, 34, 111].

Model fitting & evaluation

Since AR series models are parameter models, hyper-parameters, i.e., the order of p, d, and q, need to be determined in advance. In most applications using ARIMA model, the differencing order $d = 1$ or 2 can achieve reasonable results [111], where higher order differencing compromises the physical meaning and increases chances of over-fitting. Orders p and q can be estimated by using stationary TS properties based on the observations of ACF and PACF. Shumway & Stoffer [111] stated the following about the ACF and PACF for the AR model selection:

- TS suitable for AR(p) model is observed as ACF tails off (i.e., the ACF values become gradually smaller with the increase of the time lag) and PACF cuts off (i.e., with the increase of the lag, most of the PACF values do not exceed certain boundary values, e.g., the boundary of the confidence intervals of the PACF) after lag p.
- TS suitable for MA(q) model is observed as ACF cuts off and PACF tails off after lag q.
- TS suitable for ARMA(p, q) model is observed as both ACF and PACF tails off after lags p and q.

Based on the above, the implementation of ARMA/ARIMA can be conducted as follows: (1) compute and plot the ACF and PACF of the TS, (2) find the cut-off and tail-off lags, and then (3) empirically determinate the orders based on these lags.

Besides the manual evaluation using TS properties, there are other numerical approaches to determine the model specifications such as using *Akaike's information criterion* (AIC), Eq. 13.9 and *Bayesian information criterion* (BIC), Eq. 13.10. It is noted that the work of Nair et al. [87] and Noh et al. [90] are based on the AIC to determine the optimal order for the adopted AR models.

$$AIC = 2k - 2\log(\hat{L}) \tag{13.9}$$

$$BIC = k\log(n) - 2\log(\hat{L}) \tag{13.10}$$

where the log function in Eqs 13.9 & 13.10 is the natural logarithm function, n is the number of samples, k is the number of coefficients in the model, and $\hat{L}$ is the maximum value of the likelihood function[1] for the model.

After the fitting process, the model can be diagnosed by the *residuals check* to avoid undesired performances of over- and under-fitting. The residuals (i.e., differences between ground truth and fitted data) should share similar properties to white noise, such as *independent and identical distribution* (i.i.d), zero mean, and limited value of standard deviation (Std) (e.g., $\sigma = 2$, [22]) . Statistical methods to examine the residuals include standardization plots, quantile-quantile (QQ) plots for normality, and ACF plots for autocorrelation. However, for processing a large amount of data, manually checking each segment of the TS with these plots is inefficient, especially if there are many TS to analyze as in the case of a large instrumented structural system, e.g., a continuously monitored long span bridge with many sensors. Therefore, *Ljung-Box statistics* [70] can be applied to automatically diagnose the residuals based on the *hypothesis testing* [63]. The hypothesis testing is one traditional approach to determine the statistical relationship in the data between two interpretations, namely *null hypothesis* and *alternative hypothesis*. The former usually represents a commonly accepted fact or default assumption, and the latter is the opposite. They are mutually exclusive, i.e., if one is true, the other must be false. The decision for accepting/rejecting the null hypothesis is based on whether the p-value (calculated probability) exceeds a threshold (significant level), which is usually taken as 0.01, 0.05, or 0.10. Herein, once the p-value is high and exceeds the designated threshold, the null hypothesis is accepted which implies that the residuals check is satisfied.

13.2 ARIMA-ML Algorithm

SHM for damage detection can be considered as a classification problem within the scope of supervised learning. In this part of the dissertation, one major goal is to integrate both TS modeling and ML together to form an end-to-end ARIMA-ML algorithm/framework. Fig. 13.1 schematically illustrates the framework by first passing the raw TS data into the preprocessing module with SSND operations and stationarity check to generate the desired segments. This is followed by applying an ARMA model to act as a damage feature extraction module. It is noted that from the definition of the ARIMA model, fitting the d times differentiated segments by ARMA(p, q) models is equivalent to fitting an ARIMA(p, d, q) model. Finally, these features are fed into the classification module to run multiple ML classification algorithms with a voting mechanism for damage state identification or pattern recognition for the ultimate SHM decision-making step. The details of each module are described in the following subsections. This framework is considered as an extension of

[1]The likelihood function, $L(X;\theta)$, measures how well the model fits the data for given values of the unknown parameters. Numerically, $L(X;\theta)$ is equal to the probability of the random variable X after the parameters θ are known, refer to [100]. Accordingly, $\hat{L} = \underset{\theta}{\operatorname{argmax}}\, L(X;\theta)$.

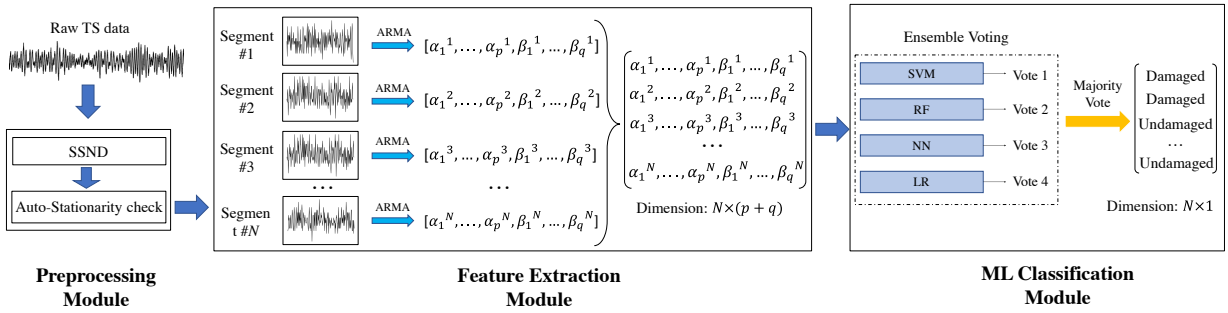


Figure 13.1: Automatic end-to-end framework for the developed ARIMA-ML algorithm

previous works. Nair et al. [87] & Noh et al. [90] used coefficient-based modeling, i.e., AR and ARMA models, for feature extraction, but here the ARIMA model is adopted to alleviate non-stationarity issues. The pipeline feeds extracted features into powerful ML classifiers, e.g., neural network (NN) according to [23]. However, the framework has an important feature applying an *ensemble technique* to make the entire framework more robust.

Smoothing-segmentation-normalization-differencing (SSND)

In signal processing, data preprocessing is an essential first step. In the framework of ARIMA-ML, firstly, MA operation (not to be confused with MA model) with a suitable window size (e.g., 2 in this study) is used to smooth the raw signals and remove some noisy data points. In addition, while adopting segmentation in the TS, it is easier to achieve the local stationarity for a short time period, because strong nonlinearity of the TS signal can be relieved by fitting local linear models for each segment. Moreover, normalization (standardization) resolves the sensor-to-sensor scale variations of the TS due to different loading, calibration, or resolution conditions. It is noted that Nair et al. [87] & Noh et al. [90] indicated the effectiveness of performing segmentation and normalization.

It is worth pointing out that the segmented TS should maintain enough information of the structural properties (e.g., stiffness) where each segment contains at least one reciprocating loading cycle, e.g., the three sample segments in Fig. 13.2. While conducting sliding window, two patterns, namely (1) *non-overlapping* and (2) *overlapping*, can be applied as discussed in the numerical experiments in Chapter 14. It is noted that the segmentation increases the amount of data especially when overlapping is adopted, which is, in general, useful for ML.

To minimize the variance of the local time period, instead of normalizing the whole signal, it is normalized within each segment. Let X_j^i be the j-th segment in the i-th record of a collection of TS signal records, the normalized signal, $\tilde{X}_j^i$, is obtained by subtracting the mean of X_j^i, i.e., $\mu_{X_j^i}$, and then dividing by its Std, i.e., $\sigma_{X_j^i}$, as follows:

$$\tilde{X}_j^i = \left(X_j^i - \mu_{X_j^i}\right) / \sigma_{X_j^i} \tag{13.11}$$

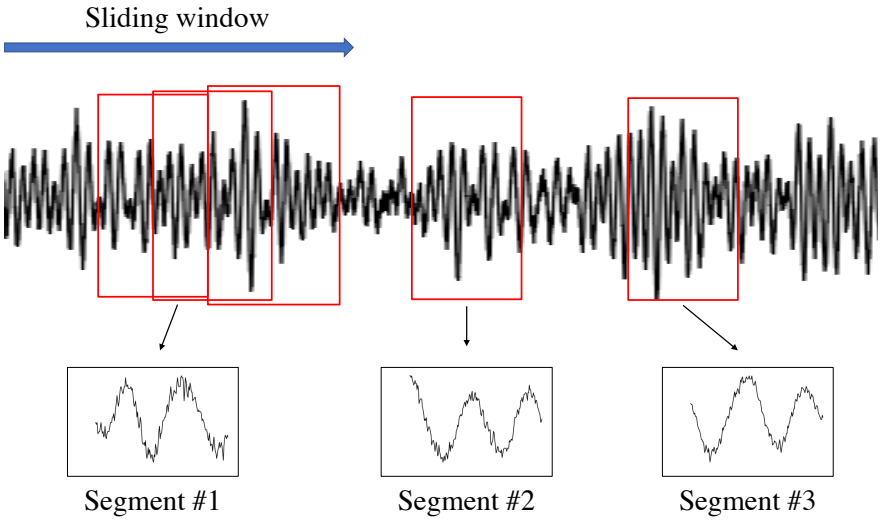


Figure 13.2: Illustration of the segmentation operation using sliding window for one sample TS signal

After normalization, d times of differencing operations are conducted for each segment. The selection of d is based on the demand (least operations to achieve stationarity) and varies between cases as discussed in a previous section.

Auto-stationarity check

Even though SSND is conducted on the raw TS, stationarity of the processed data is not guaranteed. Moreover, manually examining all segments is time-consuming and impractical. Thus, an automated filtering of the non-stationary segmentation is introduced through using the ADF test [34]. In this test, the null hypothesis is accepted by the presence of a unit root in a univariate TS signal, which represents the non-stationarity condition. On the other hand, its alternative hypothesis is test-dependent and represents for the stationarity or trend-stationarity[2]. If the p-value from the ADF test performed after SSND on a segment is larger or equal to 0.05, the null-hypothesis is accepted, which means that the segment is non-stationary and it is subsequently discarded. Similarly, if the p-value is less than 0.05, the null-hypothesis is rejected, and the alternative hypothesis is selected, which indicates that the segment satisfies the stationarity condition.

Feature extraction module

The general ML classification process is described as extracting features from the raw data, determining the mapping relationship between the features and their labels, and finally tuning

[2]If an underlying linear or nonlinear trend, e.g., a function is solely given in terms of time, can be removed while leaving a stationary process, the TS is said to be a trend-stationary [111].

the algorithms/models to obtain the best mapping function. Nair et al. [87] & Noh et al. [90] presented promising detection results of feature extraction using combinations of the first several ARMA coefficients. However, such features may not be sufficient for general structural systems, where damage patterns are not simply represented by for example using reduced cross-sections as conducted in many idealized laboratory experiments. It is known that the AR coefficients contain information about the modal frequencies and damping ratios [71], and the changes (indicative of damage) in the stiffness of the structure are accordingly reflected by the changes in these coefficients [87], which can be used for damage detection. However, for the ARIMA modeling, the differencing operation is conducted multiple times on the raw TS signals, making the physical meaning more abstract. In the ARIMA-ML, all coefficients of the AR, α_i ($i \in \{1, 2, \ldots, p\}$), and MA, β_j ($j \in \{1, 2, \ldots, q\}$), are concatenated into a $(p+q)$ damage feature vector and then several ML algorithms are applied on the whole feature vector.

ML classification module

With the extracted features, four general ML algorithms, namely (1) logistic regression (LR), (2) support vector machine (SVM), (3) random forest (RF), and (4) NN, are jointly applied for the damage classification purposes. To improve the generalization and robustness over a single algorithm, an ensemble voting classifier (EVC) is adopted. The characteristics of the EVC can be described as: (i) the final predicted results are the majority of the predictions from each individual algorithm, and (ii) the results are selected based on the "best" (highest test accuracy and smaller variance if applicable) individual algorithm performance in the cases of a tie in voting.

13.3 Summary

Concatenating the above-mentioned steps and modules describes the end-to-end framework of the ARIMA-ML algorithm, refer to Fig. 13.3 for a pseudo-code of the developed algorithm. Thus, through inputting TS signal and target TS model (e.g., AR, ARI, and ARIMA) with determined orders, the framework automatically processes and analyzes the TS, and then outputs the corresponding label, i.e., damage state or pattern. While applying the ARIMA-ML framework to the vibration-based SHM (such as the numerical experiments of a steel structure tested on a shaking table, as discussed in Chapter 14), its input is the acceleration response collected before and after a possible damaging loading scenario, e.g., due to an earthquake. It is noted that such responses can be excited from small white noise signals applied to the considered structure. The underlying assumption is that the structural damage state or pattern is determined from the damaging loading stage, and the white noise excitation with low amplitude does not cause further damage, i.e., the structure responds linearly with a degraded stiffness compared to its undamaged state. This is necessary for the accuracy of using the AR series models in vibration-based SHM applications.

ARIMA-ML Algorithm *Require*: time series $Y \in \{Y_1, Y_2, \dots, Y_n\}$ and corresponding damage states $DS \in \{DS_1, DS_2, \dots, DS_n\}$. *Define*: segmentation size L, overlapping length OL, ARIMA order (p, d, q), feature matrix F, label matrix S

function Feature Extraction
  Set j = 1 # *Count for number of segmentation*s
  for $i \leftarrow 1$ to n **do**
    Initialize $\text{index}_{\text{start}}$ and $\text{index}_{\text{end}}$
    while $\text{index}_{\text{end}} \leq \text{length}(Y_i)$ **do**
      Set $\text{index}_{\text{end}} = \text{index}_{\text{start}} + L$
      # SSND preprocessing procedure with differencing order d
      $Dseg \leftarrow \text{SSND}(Y_i[\text{index}_{\text{start}}: \text{index}_{\text{end}}],\ \text{order} = d)$
      # Auto-stationarity Check
      p-value $\leftarrow \text{ADF}(Dseg)$
      if p-value ≥ 0.05 **do** *# Check if violate stationary condition*
        Break to outer loop and **Update** $\text{index}_{\text{start}} = \text{index}_{\text{start}} + OL$
      end if
      # Feature extraction
      $[Coef_{AR}, Coef_{MA}], Residual \leftarrow \text{ARMA}(p, q)$ *# fit ARMA model with order p and q*
      # Residual Check
      p-value $\leftarrow$ Ljung-Box statistic ($Residual$)
      if p-value ≤ 0.05 **do** *# Check if violate residual assumption*
        Break to outer loop and **Update** $\text{index}_{\text{start}} = \text{index}_{\text{start}} + OL$
      end if
      Stacking $[Coef_{AR}, Coef_{MA}]$ as new row of feature matrix, F_j
      Stacking damage information to label matrix $S_j = DS_i$
      Update $\text{index}_{\text{start}} = \text{index}_{\text{start}} + OL, j = j + 1$
    end while
  end for
end function

function Classification
  Initialize $Clf_1 \leftarrow SVM, Clf_2 \leftarrow RF, Clf_3 \leftarrow NN, Clf_4 \leftarrow LR$
  if training **do**
    $Clf_5 \leftarrow EVC.fit(Data = (F, S), clf = (Clf_1, Clf_2, Clf_3, Clf_4)$
  end if
  if prediction **do**
    $\tilde{S} \leftarrow EVC.predict(Data = (F_{new}), clf = (Clf_1, Clf_2, Clf_3, Clf_4)$
  end if
end function

Figure 13.3: Pseudo-code of the ARIMA-ML algorithm

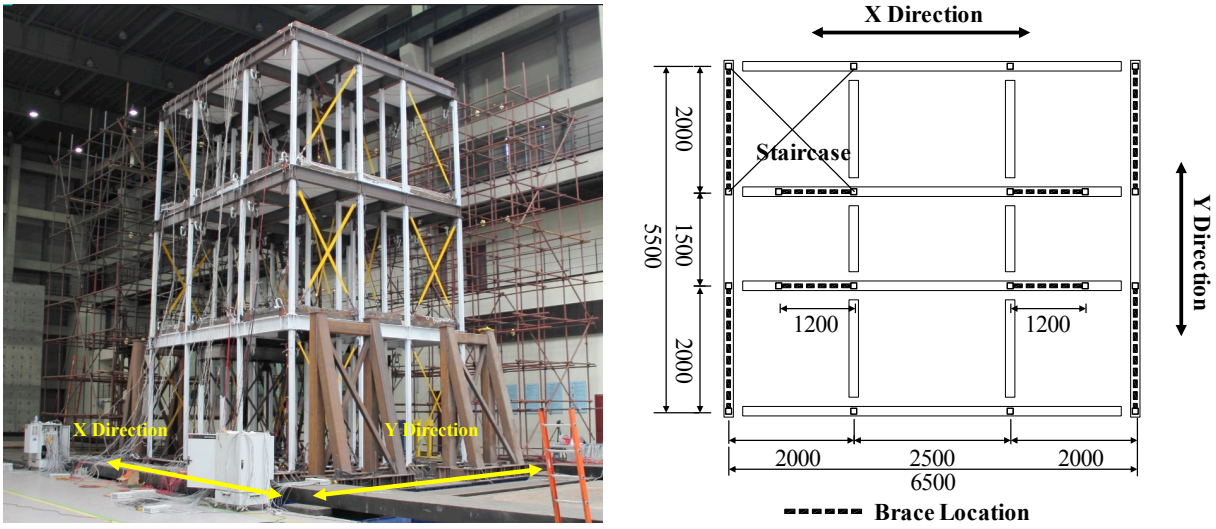
Chapter 14

ARIMA-ML for Structural Damage Recognition

In this chapter, numerical experiments are conducted to evaluate the performance of the ARIMA-ML algorithm/framework for structural damage recognition. As a data-driven approach, for the purpose of validation, the used data are from full-scale shaking table experiments of a steel braced frame structure representing realistic practical construction conditions. The conducted study in this chapter for ML-centric vibration-based SHM is complementary to a prior human expert-centric investigation by the researchers (structural engineers) who conducted the physical experiments, as documented in [18]. Moreover, the presented study builds on related research focused on ARMA modeling and ML classification as documented in [80, 87, 90]. Moreover, three independent classification experiments were designed, namely: (1) general damage state detection, (2) local damage state detection, and (3) local damage pattern detection, to evaluate the accuracy and robustness of the ARIMA-ML framework for different recognition purposes. In addition, to better understand the developed models, the importance of the features was also discussed along with the obtained classification results.

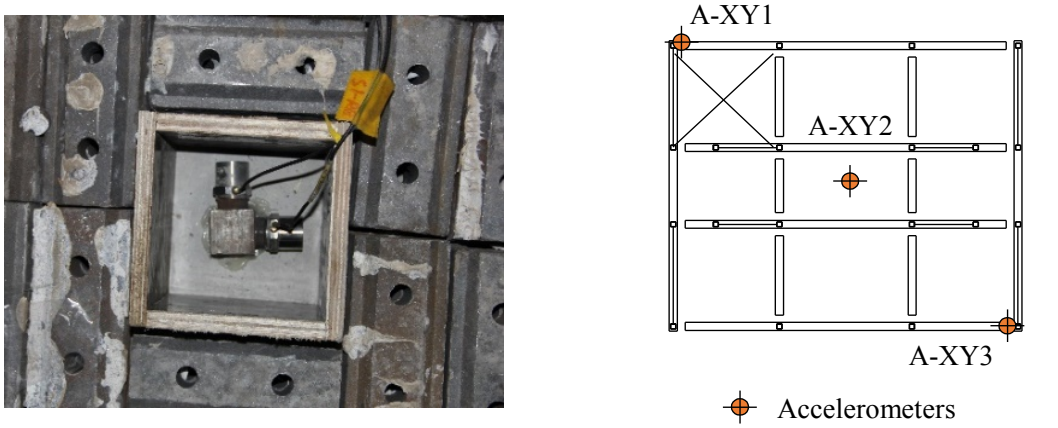
14.1 Experimental Data Source

In this study, raw data were obtained from full-scale shaking table experiments conducted in Tongji University, China, on a 3-story, 3-bay, tension-only concentrically braced beam-through frame (TCBBF), Fig. 14.1, under different earthquake hazard levels [18]. This experiment is selected to demonstrate the application of the ARIMA-ML algorithm to a general steel structure. The test structure had a total height of 10.02 m (story height = 3.34 m). The sensors, Fig. 14.2, included accelerometers installed at the center and two corners of every floor to measure the response in two horizontal directions. The structure was shaken using natural and artificial ground motions for frequently occurring earthquake (FOE), design basis earthquake (DBE), and maximum considered earthquake (MCE). These levels were defined



(a) General view (b) Plan layout

Figure 14.1: 3-story tension-only concentrically braced beam-through frame (TCBBF) [18]



(a) 2D accelerometers (b) Typical floor arrangement

Figure 14.2: Sensors layout [18]

using peak ground acceleration (PGA) according to seismic intensity 7 in the Chinese code for seismic design of buildings [20], namely $0.035g$, $0.1g$, and $0.22g$, respectively, where g is the acceleration of gravity. To evaluate the structural damage state, low amplitude and non-damaging white noise signals were applied before and after the different earthquake loading levels.

During the experiments, no fracture occurred in the structural members, i.e., the beams, columns, and the beam-column joints. Therefore, the frames remained elastic in most of the loading cases. Only the braces yielded and loosened after the earthquake loading, Fig. 14.3. Hence, there were two damage patterns observed for each floor: (1) loosening of partial



(a) Braces in X direction



(b) Braces in Y direction

Figure 14.3: Loosening of the braces [18]

braces, and (2) loosening of all braces. Both patterns were well-recorded and used for data labeling for the recognition tasks, refer to Table 14.1. For simplicity during the ARIMA-ML algorithm validation, only the X direction data were used, because more significant damage was observed in this direction.

14.2 Experimental Objectives

In the validation experiments, raw data are signals of structural acceleration response under white noise, collected from sensors on each floor, which are in the format of TS. Considering only the X direction and the rigid diaphragm assumption [19, 82], the steel structure can be idealized as a 2D multi-degree-of-freedom (MDOF) system with three translational DOFs per each floor, Fig. 14.4, and the detection accuracy is restricted to the floor level. Each signal represents floor response under an earthquake loading and was labeled with the damage state and the pattern of the floor level as determined by observations and brace strain information according to the test records. Sensors in different locations of one floor share the same label due to the rigid diaphragm assumption.

Damage detection and localization are of major interests in SHM. Thus, three recognition tasks are pursued herein: (1) general damage state detection, (2) local damage state detection, and (3) local damage pattern recognition.

In the general damage state detection task, the global health condition of the whole structure is assumed to be independent of the damage location. In other words, occurrence of damage in the structure denotes a damaged state (a binary classification problem). For this purpose, all signals were mixed together with labels of either undamaged state or damaged state ignoring the floor information. This can be considered as one idealization of the raw structure as a 2D single-degree-of-freedom (SDOF) if only considering the X direction as an example, Fig. 14.4. In the local damage detection task, more fine-grained detection is conducted to locate the damage based on the floor level. Only signals collected from the same

Table 14.1: Structural information and damage patterns recorded under earthquake loading (UD: undamaged state, PL: partial braces loosened, and AL: all braces loosened)

Hazard level	**Record**	**PGA (g)**		**Fundamental period (sec)**	**Damage patterns**
		X	**Y**		
FOE	El Centro-X[1]	0.035	0.03	0.56	No damage
	El Centro-Y	0.03	0.035	0.56	No damage
	SHW2-X[2]	0.035	0.03	0.56	No damage
	Kobe-X[3]	0.035	0.03	0.56	No damage
	Kobe-Y	0.03	0.035	0.56	No damage
DBE	El Centro-X	0.1	0.085	0.56	No damage
	El Centro-Y	0.085	0.1	0.56	No damage
	SHW2-X	0.1	-	0.56	No damage
	Kobe-X	0.1	0.085	0.67	UD in floor 1 & 2; AL in floor 3
MCE	El Centro-X	0.22	0.187	0.74	PL in floor 1 & 2; AL in floor 3
	El Centro-Y	0.187	0.22	0.81	PL in floor 1 & 2; AL in floor 3
	SHW2-X	0.22	-	1.13	PL in floor 1 & 2; AL in floor 3
	SHW2-Y	-	0.22	1.13	PL in floor 1 & 2; AL in floor 3
	Kobe-X	0.22	0.187	1.36	AL in all floors
	Kobe-Y	0.187	0.22	1.36	AL in all floors
A-DBE[4]	El Centro-X	0.1	0.085	1.36	AL in all floors
	El Centro-Y	0.085	0.1	1.36	AL in all floors

[1] El Centro record from the 1940 California Imperial Valley earthquake.
[2] Shanghai artificial accelerogram.
[3] JMA Kobe record from the 1995 Kobe earthquake.
[4] Aftershock at the DBE level applied after the MCE level of Kobe-X earthquake.

floor were mixed along with their damage state, and the damage check was performed floor by floor. Therefore, this task is three pairs of a binary classification problem. Finally, in the local damage pattern recognition task, the development of damage in the tested steel frame was further investigated. As mentioned above, three different patterns based on the floor level were assigned: undamaged (UD), partial braces loosened (PL), and all braces loosened (AL), which is a three-class classification problem.

As introduced in Chapter 2, unlike three of the ML classifiers used in ARIMA-ML, i.e., support vector machine (SVM), logistic regression (LR), and neural network (NN), the random forest (RF) algorithm performs the classification by splitting the data into subsets that most heavily belong to one specific class based on data features weighted as important if they make the subsets distinguish the classes better. Thus, the feature importance (FI) is a good indicator for the evaluation and for better understanding of how the RF conducts

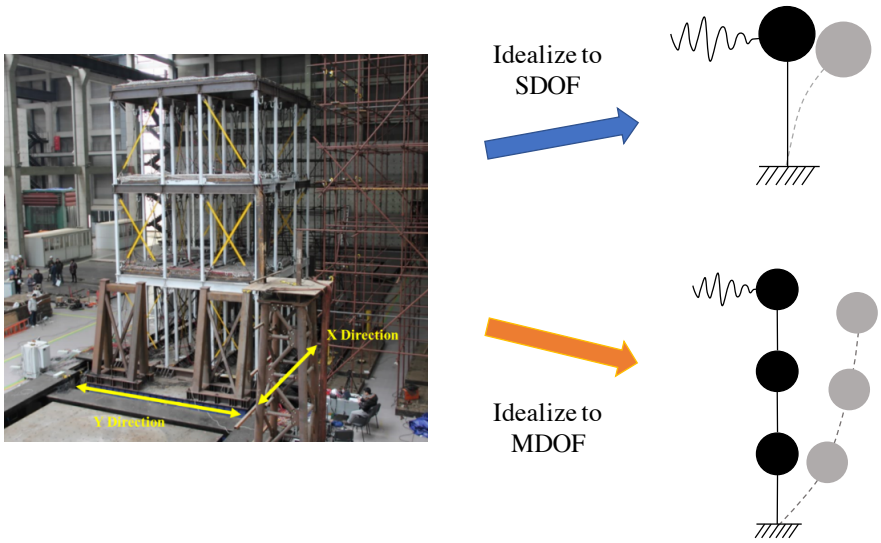


Figure 14.4: Structural idealization of the tested steel structure

the classification via features. FI is calculated as the decrease in node impurity (e.g., Gini Impurity introduced in Section 2.1) weighted by the probability of reaching that node. The node probability can be calculated by the number of data that reach the node, divided by the total number of data. The higher the value, the more important the feature is. For more details related to the FI, refer to [10]. Moreover, since all the ARIMA coefficients are concatenated as features, FI provides guidance for the importance of each coefficient in the classification.

14.3 Exploratory Data Analysis

Because the ARIMA model is a parametric model, where the orders of AR, MA, and differencing need to be pre-determined first and then tuned for the best fitting results. In many statistics applications, *auto-arima*[1] is a useful tool to automatically determine the best ARIMA model. However, considering the difference between each TS segment among shaking table test records and the necessity to keep the same length of the feature vector for the ML module, the same number of AR and MA parameters should be generated from the same model. Thus, such constraints make the auto-arima inapplicable here because the auto-arima may fit different ARIMA models for each individual segment and may also over-estimate the local characteristics of the segments leading to loss of generality. Therefore, using a fixed-order ARIMA model that can generate satisfactory performance in fitting most of the TS segments is considered in this study.

In order to determine the orders of the different parts of the adopted ARIMA model, an exploratory data analysis (EDA) for all segments was configured and conducted as follows:

[1] `https://www.rdocumentation.org/packages/forecast/versions/8.5/topics/auto.arima`.

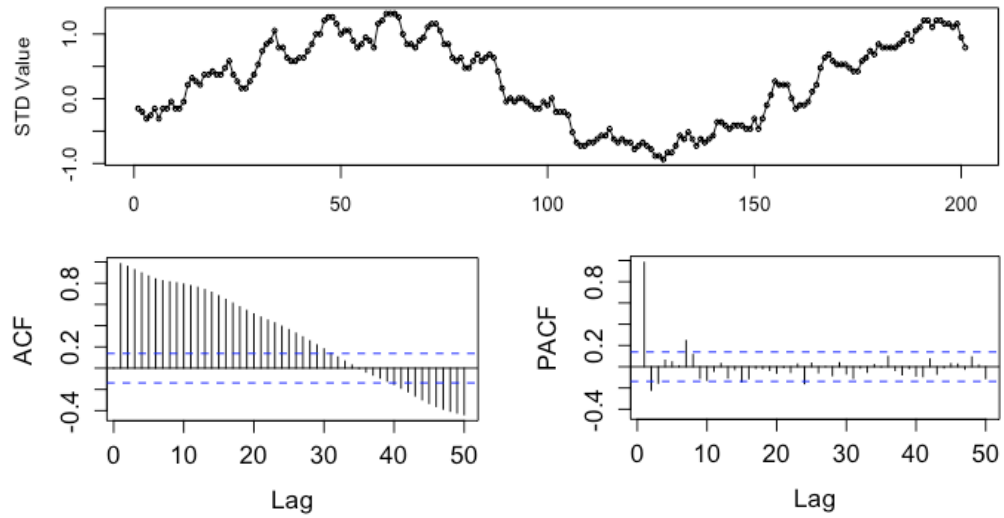
1. Check the ACF and PACF to narrow the range of the orders.
2. Evaluate the sub-models via the BIC values to select candidate models.
3. Evaluate the average AIC values for each candidate model.
4. Perform residual diagnosis for each candidate model.

TS properties for the order range determination

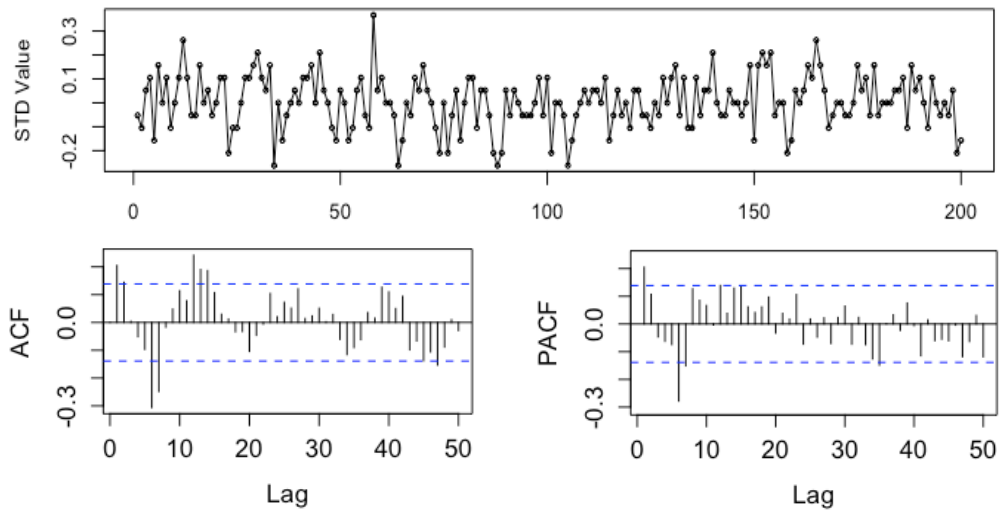
Firstly, several preliminary investigations were conducted via plotting the ACF and PACF of the TS segments with SSN and SSND, Fig. 14.5. Even after SSN, Fig. 14.5a shows certain trends in the ACF and its values diminish slowly, indicating non-stationarity. By performing differencing once after SSN, the performance in both ACF and PACF plots significantly improves, where the ACF diminishes rapidly within the shown dashed lines with less than 15 lags and the PACF cuts off certain number of lags, Fig. 14.5b. This observation roughly meets the property of stationary condition and indicates that the differencing order d can be taken as 1. The stationarity of the TS with the ADF test was also evaluated, and p-values of 0.78 and 0.01 were achieved for the cases of SSN and SSND, respectively. The former one significantly exceeds 0.05, which accepts the null hypothesis (non-stationary), but the latter one rejects the null hypothesis and accepts the alternative hypothesis (stationarity). After examining several plots of different segments, it is concluded that stationarity was difficult to achieve for the collected signals if only SSN was used. However, one-time differencing significantly relieved this issue, supporting the need to use ARIMA models instead of the ARMA series models in the current application. It is noted that if one-time differencing still cannot satisfy the stationary condition, the segment would be discarded. Moreover, from the same plots, it is concluded that lags 6 or 7 can be thought of as the cut lag. As mentioned above, a fixed-order ARIMA needs to be determined for maintaining the fixed feature dimension (i.e., $p + q$). For better generalization to fit most segments well using one fixed-order model, a fixed order p for the AR can be taken as 6 or 7. In the ACF plots, autocorrelation diminished, but it was difficult to decide from where the correlation tails off. According to Shumway & Stoffer [111], to avoid overfitting, here q can be empirically taken as 6. In summary, using the properties of the TS from this preliminary analysis, the model specification and order can be selected.

BIC-based candidate models selection

BIC values were explored for sub-models of ARIMA$(p, 1, q)$ with different combinations of $p = 0$ to 7 and $q = 0$ to 6, Fig. 14.6. Each row represents one specific sub-model with different combinations of the model parameters for the AR (p) and MA (q) terms, represented by the gray color depth in each column. For example, in Fig. 14.6a, the first row represents the sub-model of ARIMA$(7, 1, 6)$ with BIC value -590, and its parameters are related to TS X-lag (AR terms) 1, 2, 3, & 7 and error-lag (MA terms) 2, 4, & 6. Better models are in the top



(a) TS segment after SSN without differencing



(b) TS segment after SSND

Figure 14.5: Standardized (STD) value, ACF, and PACF of one sample TS segment with different preprocessing (Dashed lines: approximate 95% confidence intervals of the white noise results & vertical lines: values beyond which autocorrelations are (statistically) significantly different from zero [46]).

Table 14.2: Average AIC of three candidate models

Pattern	**ARI(7,1)**	**ARIMA(6,1,6)**	**ARIMA(7,1,6)**
Non-OL	-417.20	-450.72	-451.63
OL	-418.24	-450.73	-451.60

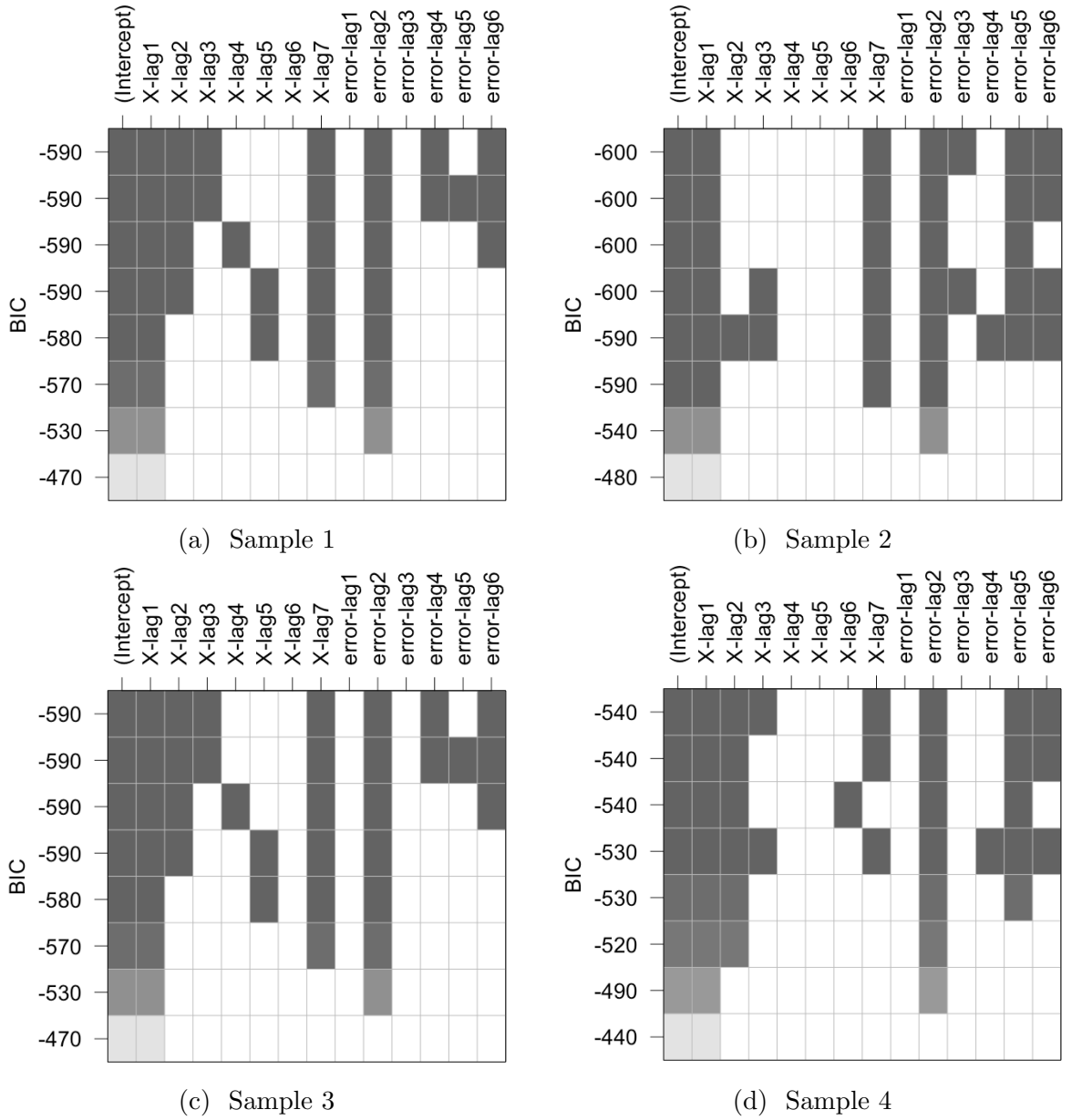
rows with lower BIC values. It is noted that there were multiple sub-models sharing the same BIC values, i.e., top four rows in Fig. 14.6a. Although they show contributions from different combinations of the lower order lags, the higher orders TS X-lag 7 and also to a lesser extent the error-lag 6 are present in most cases. Based on results of other segments, Figs 14.6d & 14.6c, the influence of TS X-lag 7 and error-lags 2 and 6 in dominating the BIC performance is detected. This is an evidence to incorporate higher orders in the ARIMA models. From the observations that PACF cuts off from lag 6 or 7 and ACF tails off simultaneously and from suggestions in Cryer & Kellet [22], ARI$(7, 1)$, ARIMA$(6, 1, 6)$, and ARIMA$(7, 1, 6)$ were selected as candidate models in the present application.

AIC-based candidate models evaluation

The AIC values were examined for more insight into the candidate models. Since segmentation operation was conducted, which caused the AIC value to vary from one segment to another, the average AIC values were computed from all segments of each model. To explore the influence of overlapping in the SSND procedure, the average AIC was also evaluated for both non-overlapping (Non-OL) and overlapping (OL) patterns. Table 14.2 shows that a sliding window with OL does not affect the average AIC, indicating a stable performance. It is also shown that the ARIMA$(6, 1, 5)$ and the ARIMA$(7, 1, 6)$ achieved similar average AIC with the latter slightly lower (algebraically) but both models had significantly lower values than that of the simpler ARI$(7, 1)$ model. Note that, in contrast with the ARIMA model, the ARI model only includes the AR part together with the differencing.

Model residual diagnosis

After fitting the three candidate models selected above, the residual diagnosis were performed. From the general results, the three models achieved similar performance, Fig. 14.7. Only few outliers fall outside the 2 standardized residuals, and most points were within 1, indicating normal distribution. The correlation of the residuals using the ACF diminished and remained bounded by the white noise boundary (dash lines) indicating a low relationship of residuals. From the quantile-quantile (QQ)-plot, most points aligned along one line, which presents i.i.d. Hence, all three models satisfied the residual assumption where the ARIMA$(6, 1, 6)$ & ARIMA$(7, 1, 6)$ models were slightly superior than the ARI$(7, 1)$ model with fewer outliers in the ACF, STD value, and QQ-plots.



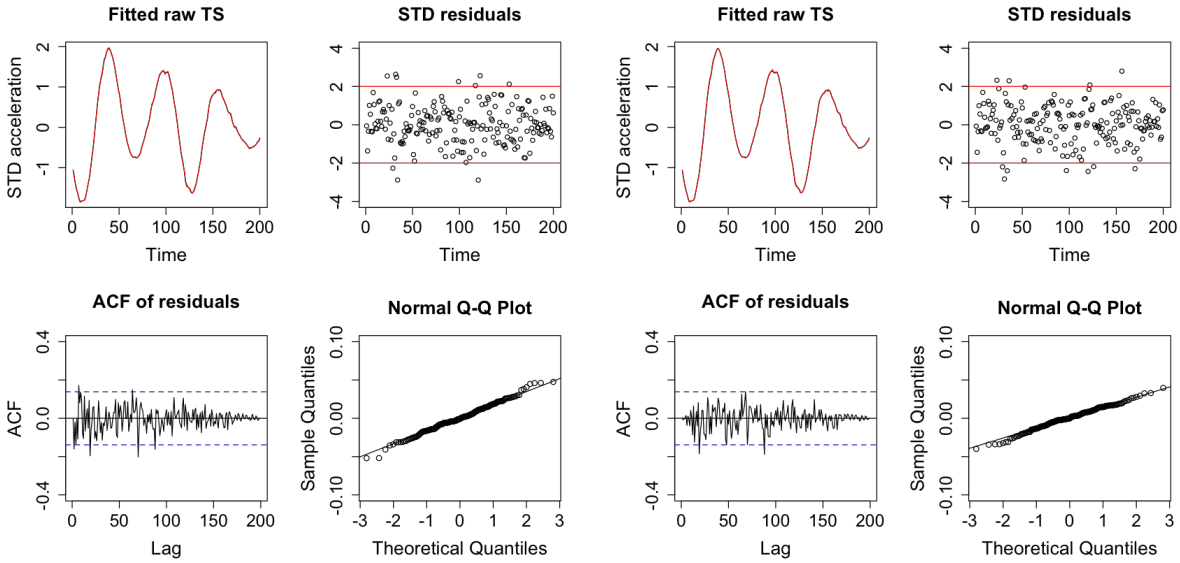
(a) Sample 1

(b) Sample 2

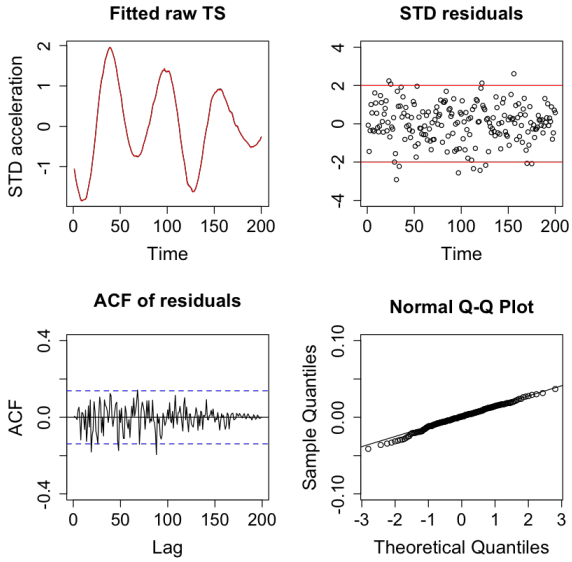
(c) Sample 3

(d) Sample 4

Figure 14.6: BIC values for sub-models computed on four sample TS segments. Horizontal axis labels the TS X-lag i (X_{t-i}) and error-lag j (ϵ_{t-j}), as expressed in Eq. 13.8, contributing to order i AR (α_i) and order j MA (β_j) terms, respectively. Intercept is the bias term in the regression. Vertical axis is the BIC computed for each sub-model. The darker the color and the lower the BIC value, the better the model.



(a) ARI$(7,1)$ (b) ARIMA$(6,1,6)$



(c) ARIMA$(7,1,6)$

Figure 14.7: Residual diagnosis for three candidate models

Table 14.3: Number of segments generated by different models (undamaged/damaged)

Pattern	**ARI(7,1)**	**ARIMA(6,1,6)**	**ARIMA(7,1,6)**
Non-OL	9,955 (4,096/5,859)	9,643 (3,944/5,699)	9,603 (3,915/5,688)
OL	19,865 (8,206/11,659)	19,197 (7,833/11,364)	19,201 (7,823/11,378)

14.4 Experimental Setup

As for the major objectives of the numerical study using the ARIMA-ML framework presented in this chapter, computer experiments were conducted for the previously discussed three recognition tasks. Even though prior knowledge on the superiority of the three candidate models was obtained based on the EDA, in order to explore the influence of the ARIMA modeling towards the classification tasks, *all candidate models* were tested and compared for these experiments. Details about the experimental settings are presented below.

As for dataset generation, while processing the TS by SSND (Fig. 13.1), according to the sampling frequency (256 Hz) of the sensors used in the shaking table experiments, 200 data points as the segmentation size was considered sufficient in this study. In other words, the segmentation window size was taken as 200. Because there existed segment selection operations in the ARIMA-ML framework, Fig. 13.3, the number of segments used for training and testing in each candidate model might be different, which resulted in different sizes of the datasets used for the different candidate models. To further investigate the influence of overlapping in the sliding window, both Non-OL and OL patterns were considered for each model. From the previously determined segmentation window size of 200, the OL size was assigned as 100. Moreover, due to different model complexity and constraints, different models generated different number of segments. Accordingly, the segments generated by OL were about twice of those generated by Non-OL, refer to Table 14.3. Thus, the relatively simple ARI$(7, 1)$ model had more data due to less modeling constraints.

While conducting ML, only test accuracy (overall accuracy, Eq. 2.7) is presented and discussed here. This accuracy indicates the generalization of the predictive classifier. The k-fold cross-validation was applied and used for tuning the hyper-parameters of the classifier using different combinations. Similar to that introduced in Section 2.1, in the cross-validation, all data were randomly and equally split into k folds, and at each time, $k-1$ folds with a new combination of folds were used as a training set and the remaining k-th fold was assigned as a test set until each fold had been tested once. In the following experiments, $k = 5$ was used. Multiple runs in cross-validation determined the relatively optimal setting for each classification algorithm, Table 14.4. Moreover, the cross-validation allowed estimating the mean and standard deviation (Std) from all cases to compare the models by their respective confidence intervals. In addition, as for the feature analysis and for better comparison, the range of FI for each feature was normalized to be $(0, 1)$ and all the FI values should sum up to 1.

Table 14.4: Implementation details of the four ML and EVC classification algorithms

Algorithm	Details
SVM	1. Use kernel-SVM with Gaussian kernel 2. Use hyper-parameter $C = 100$ (Eq. 2.23)
NN	1. Build a 2-layers network 2. Neurons in first & second layers are 200 & 100, respectively 3. Use ReLu as an activation function
RF	1. Use Gini impurity as a split criterion (Eq. 2.47) 2. Use 250 trees as the estimator and then average their predictions
LR	1. Add L2 regularization term [38] 2. Use regularization parameter $C = 100$
EVC	1. Ensemble of the above four individual algorithms with their optimal settings 2. Use a hard-voting mechanism

Table 14.5: Accuracy of the general damage state detection for different candidate models

Model	Pattern	SVM	NN	RF	LR	EVC
ARI$(7,1)$	Non-OL	95%±6%	95%±5%	94%±4%	93%±8%	95%±4%
	OL	95%±7%	95%±6%	95%±4%	93%±9%	95%±5%
ARIMA$(6,1,6)$	Non-OL	93%±7%	93%±6%	89%±7%	89%±5%	93%±7%
	OL	94%±6%	93%±6%	90%±7%	89%±4%	94%±6%
ARIMA$(7,1,6)$	Non-OL	93%±8%	94%±6%	90%±7%	91%±3%	93%±6%
	OL	94%±7%	93%±8%	90%±6%	91%±3%	94%±6%

14.5 Experimental Results & Discussions

General damage state detection

The robustness of the ARIMA-ML in distinguishing the damage state from one TS signal without consideration of the damage location was firstly explored. From the results in Table 14.5 and Fig. 14.8, it is observed that there was no significant difference between Non-OL and OL. Accordingly, the generated data without OL (about 10,000), were thought to contain sufficient information to accurately train a classifier for general damage detection. Therefore, it is believed that the ARIMA-ML algorithm is practical in the current application, because it does not require a very large dataset.

Among the different individual ML classifiers, SVM and NN achieved the highest accuracy (93∼95% for all candidate models), while RF and LR were slightly lower by about 1∼2%. Moreover, the ensemble method EVC indicated a stable and robust performance, which was typically at least as good as the best individual classifier [10, 13] and corresponded to

less variation (tighter confidence interval) between multiple runs. According to the overall accuracy of the EVC listed in Table 14.5, the three candidate models achieved similar performance where the ARI$(7, 1)$ was slightly better than the other two.

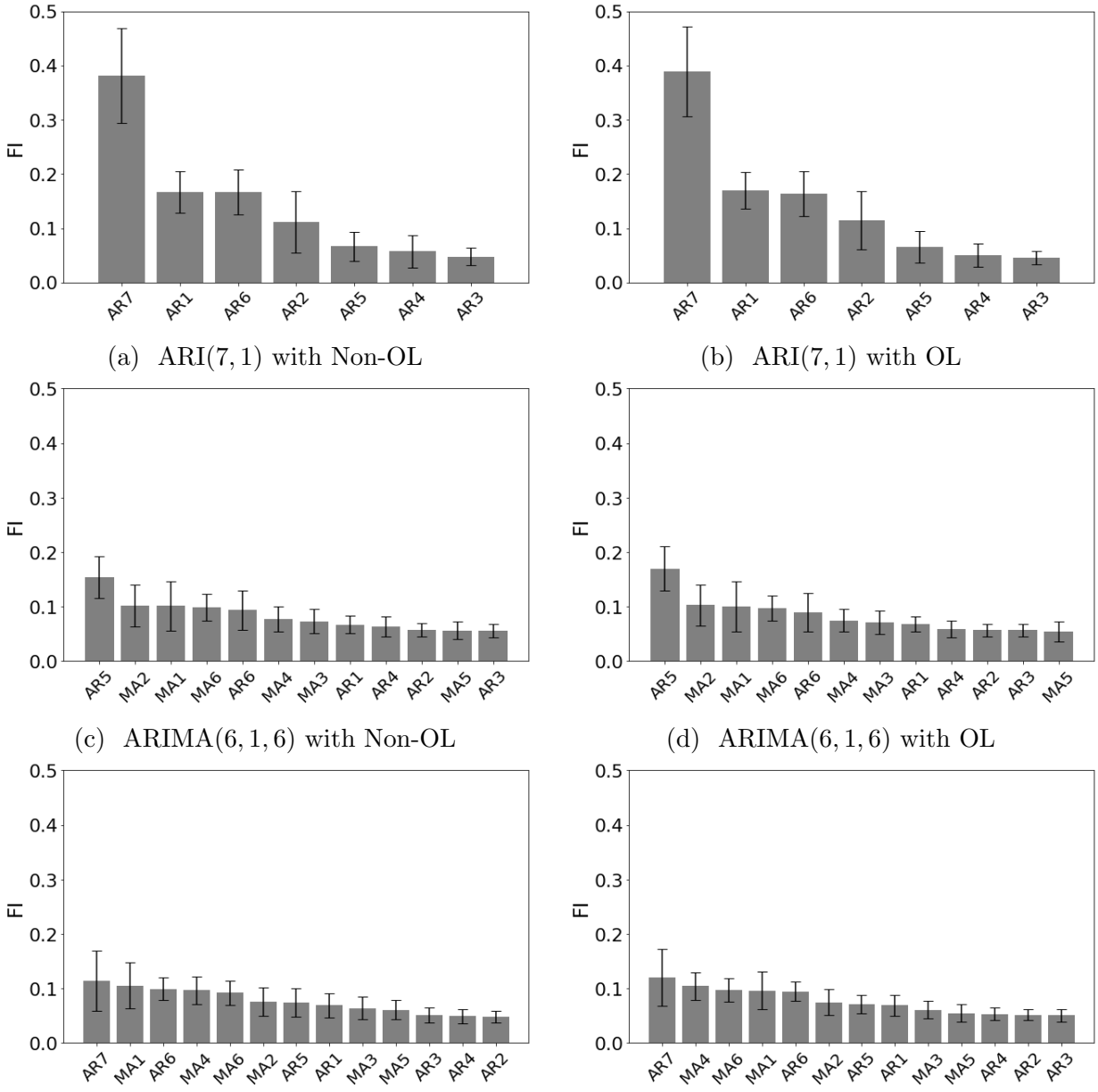
From the shaking table tests, Table 14.1, the natural period of the tested structure varied from 0.56 sec. (undamaged) to 1.36 sec. (damaged). It is noted that the floor information was not considered in this case. In other words, the whole structure was treated as a SDOF system instead of a MDOF one where damage can be detected as changes in its global stiffness and natural period, Fig. 14.4. From the results, the ML classifiers were shown to be sensitive to the changes in the ARIMA coefficients corresponding to the changes in the first natural period. This is consistent with previously published findings of high correlations with the AR coefficients using ARMA models [23, 87].

Evaluating the FI of the RF model revealed useful information about the feature selection. It was also shown to be reasonable to adopt the RF for feature analysis in this experiment since it achieved classification accuracy of 90% or higher. Fig. 14.8 reports the relative importance of each feature with respect to all features. The gray bars are the mean values and the thin lines (whiskers) are the deviations due to multiple trees used in the RF, where the FI in RF was computed from averaging the FI values of all trees [10]. From Fig. 14.8, several of the highest order coefficients played a very importance role in the classification, i.e., AR order 7 (FI ≈ 0.38) in ARI, AR order 5 & 6 (FI ≈ 0.16 & 0.10, respectively) in ARIMA$(6, 1, 6)$, and AR order 6 (FI ≈ 0.12) in ARIMA$(7, 1, 6)$. Moreover, since the coefficients were obtained from the ARIMA fitting, Eq. 13.8, the correlations between AR and MA coefficients were almost eliminated, reducing the drawback of FI (It is known that the FI performed worse and less informative in evaluating the correlated features [10, 40]). It is evident that higher order AR coefficients contained more essential information, which was sensitive to the damage detection task and some MA coefficients had significant contributes, e.g., MA 1, 2, 4, & 6 in Figs 14.8c to 14.8f. In the ARI model, the highest order AR coefficient had a dominating effect in Figs 14.8a & 14.8b. On the other hand, in the ARIMA model, the higher order AR contribution still existed but this contribution was reduced due to the MA order contributions, and the influence of all coefficients was more uniform than that for the AR-only model.

Local damage state detection

In this experiment, the local detection state was based on the floor level where only TS signals collected on the same floor were trained and tested. Due to similar performance of Non-OL and OL and for brevity, the presented results of the computer experiments in this section and the subsequent one are those from the OL pattern only.

From Table 14.6, the local detection accuracy increased to about 96%$\sim$98% in the best case, compared to that for the general damage detection (93%$\sim$95% from Table 14.5), as discussed in the previous section. This is attributed to the fact that signals from the same floor have similar properties, leading to lower variance. Moreover, EVC achieved the best mean accuracy with lowest deviation compared to the four individual algorithms in all cases (except in ARI$(7, 1)$ for the second floor with RF where the 3% deviation of the EVC was



(a) ARI$(7,1)$ with Non-OL

(b) ARI$(7,1)$ with OL

(c) ARIMA$(6,1,6)$ with Non-OL

(d) ARIMA$(6,1,6)$ with OL

(e) ARIMA$(7,1,6)$ with Non-OL

(f) ARIMA$(7,1,6)$ with OL

Figure 14.8: FI of three models for different sliding window patterns using RF. Horizontal axis shows all features used for classification in a decreasing trend ordered by the value of FI. Vertical axis stands for FI for each feature normalized by its sum for all the features, where the higher the value, the more important the feature.

Table 14.6: Accuracy of the local damage state detection for different candidate models

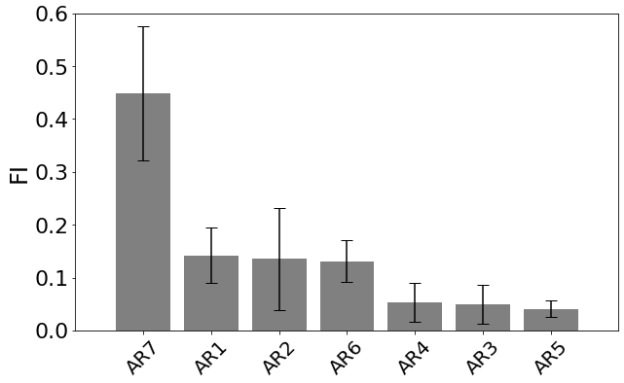
Model	Floor	SVM	NN	RF	LR	EVC
ARI(7, 1)	1	97%±3%	97%±3%	95%±3%	96%±4%	97%±3%
	2	96%±4%	96%±4%	96%±2%	91%±10%	96%±3%
	3	98%±1%	98%±1%	97%±1%	98%±1%	98%±1%
ARIMA(6, 1, 6)	1	95%±3%	94%±4%	90%±5%	91%±3%	95%±2%
	2	94%±4%	94%±5%	90%±3%	90%±4%	94%±4%
	3	98%±1%	97%±1%	94%±1%	97%±2%	98%±1%
ARIMA(7, 1, 6)	1	96%±2%	96%±2%	91%±3%	93%±3%	96%±2%
	2	94%±3%	94%±4%	90%±4%	90%±6%	94%±3%
	3	98%±1%	98%±1%	96%±1%	94%±1%	98%±1%

larger than the reported smaller deviation of 2% in this case), indicating again the stable and robust performance of the EVC. At the floor level, ARI(7, 1) model still slightly outperformed the other two complex models especially for floors 1 & 2, but all three models had the same performance for floor 3 with EVC (98%±1%). This is attributed to more significant local stiffness change of floor 3 due to brace loosening where the effective area of the braces for floor 3 was less than half of that for the other floors, Fig. 14.1a. From these results, it can be inferred that ARIMA coefficients can also capture the local stiffness changes between floors. In addition, the FI for each model based on the different floors, Figs 14.9 to 14.11, present a similar trend to that for the general damage detection case, as discussed in the previous section. Thus, this experiment, similar to the previous one, also indicated the stability and robustness of the proposed ARIMA-ML algorithm.

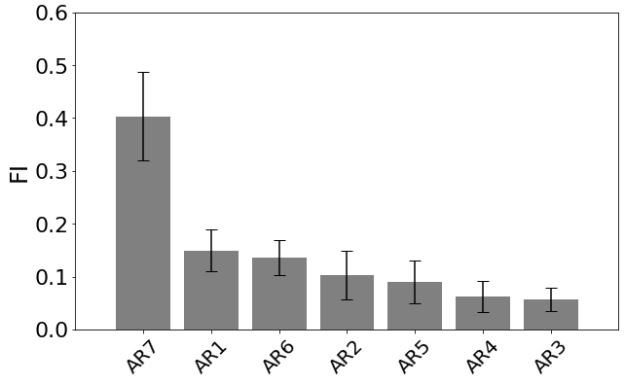
Local damage pattern recognition

Recognizing the damage pattern of the structure is important in post-disaster decision-making of the SHM process. The design of the test frame had a single brace in each of the two exterior bays (Y-direction) of the $3rd$ floor, Fig. 14.1a, which led to its fragility under earthquake excitation. As a result, the $3rd$ floor was labeled with either undamaged or all braces loosened but did not experience the partial brace damage pattern after any of the applied earthquake loading cases during the shaking table tests, Table 14.1. Thus, only data collected from floors 1 and 2 were evaluated in this pattern recognition experiment.

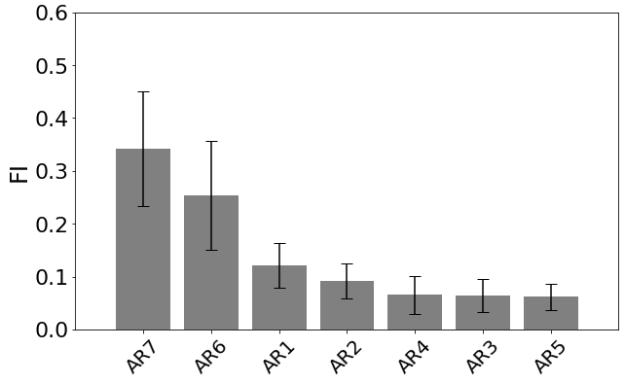
From the previous two experiments, it was shown that the ARIMA-ML algorithm can accurately distinguish between damaged and undamaged states. However, in the present experiment with increased difficulty of the task, i.e., more classes to be classified, Table 14.7 shows that the performance of all models slightly decreased, especially in the complex models with RF and LR classifiers where about 6%∼8% reductions in the accuracy were observed compared to the SVM, NN, and EVC. In general, EVC still provided the best accuracy of



(a) Floor 1

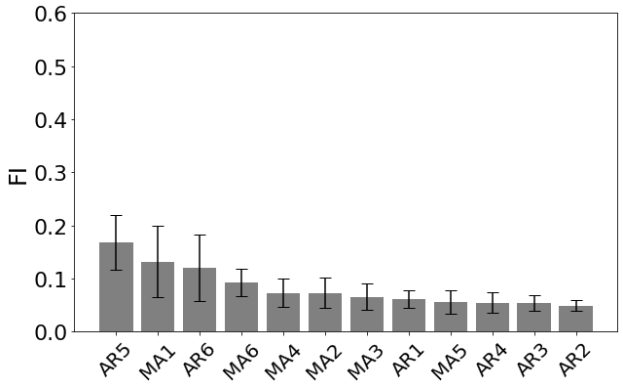


(b) Floor 2

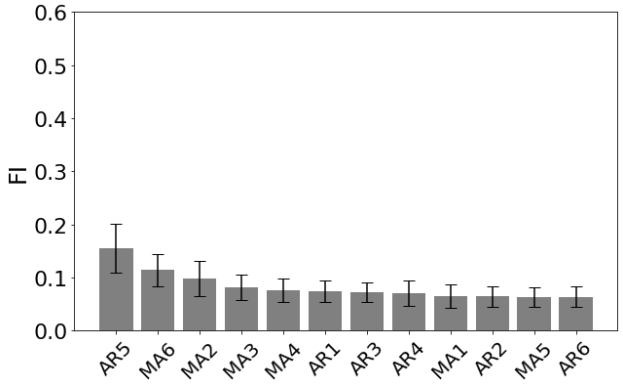


(c) Floor 3

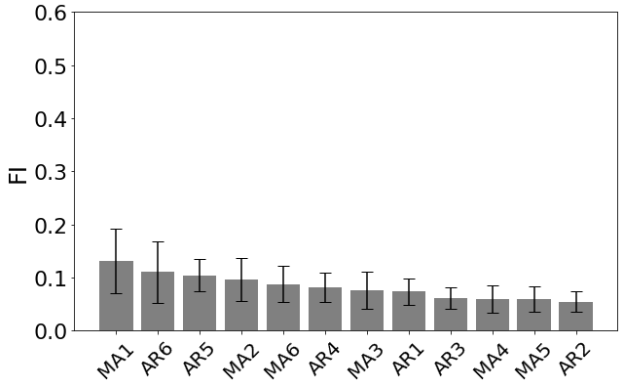
Figure 14.9: FI of ARI$(7, 1)$ in local damage state detection



(a) Floor 1

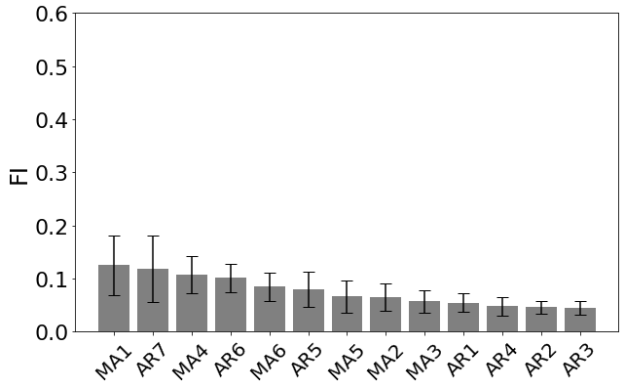


(b) Floor 2

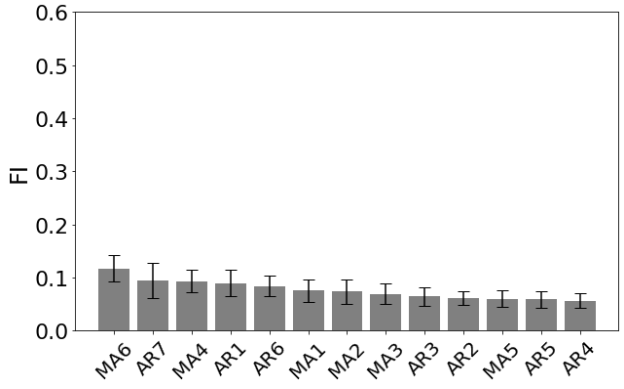


(c) Floor 3

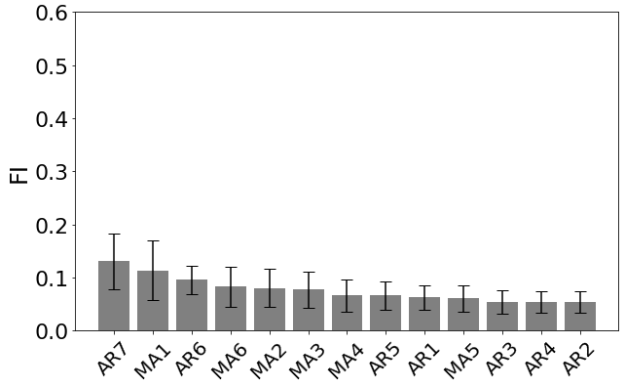
Figure 14.10: FI of ARIMA$(6, 1, 6)$ in local damage state detection



(a) Floor 1



(b) Floor 2



(c) Floor 3

Figure 14.11: FI of ARIMA$(7, 1, 6)$ in local damage state detection

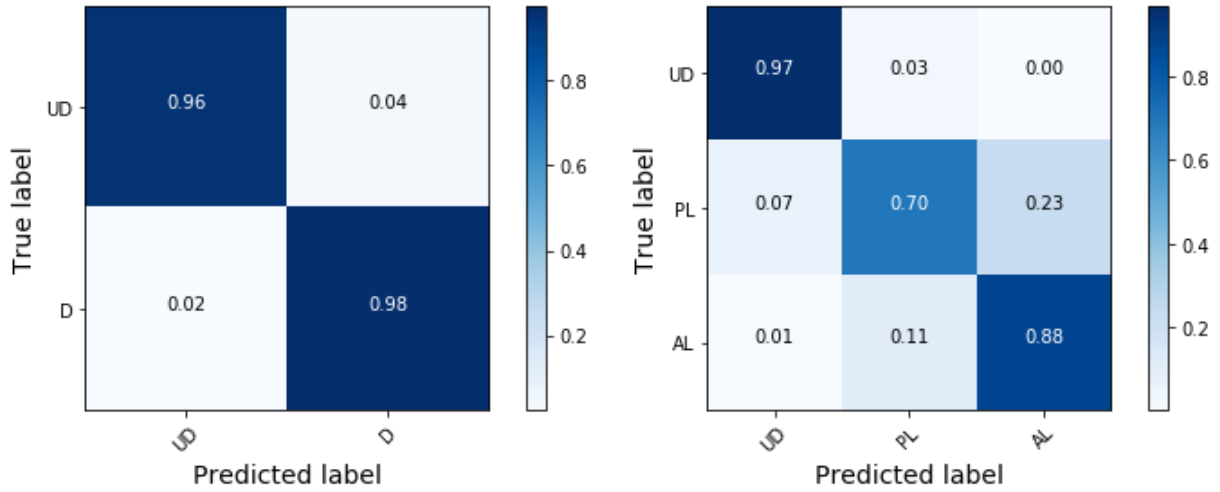
Table 14.7: Accuracy of the local damage pattern recognition for different candidate models

Model	Floor #	SVM	NN	RF	LR	EVC
ARI(7, 1)	1	87%±1%	86%±2%	86%±2%	84%±1%	87%±1%
	2	87%±3%	87%±3%	88%±3%	79%±3%	88%±2%
ARIMA(6, 1, 6)	1	84%±1%	83%±1%	74%±3%	77%±1%	84%±1%
	2	85%±3%	84%±4%	79%±4%	75%±6%	85%±2%
ARIMA(7, 1, 6)	1	85%±2%	85%±2%	77%±3%	80%±3%	85%±2%
	2	85%±2%	85%±2%	79%±1%	75%±2%	85%±2%

85%∼87% with 2% or less deviations in all cases. ARI(7, 1) with EVC again presented the best performance with the highest accuracy and lowest variance. Even though the results were lower than those in the previous two experiments, they were still much higher than a random guess of 33% for the present three-class classification problem.

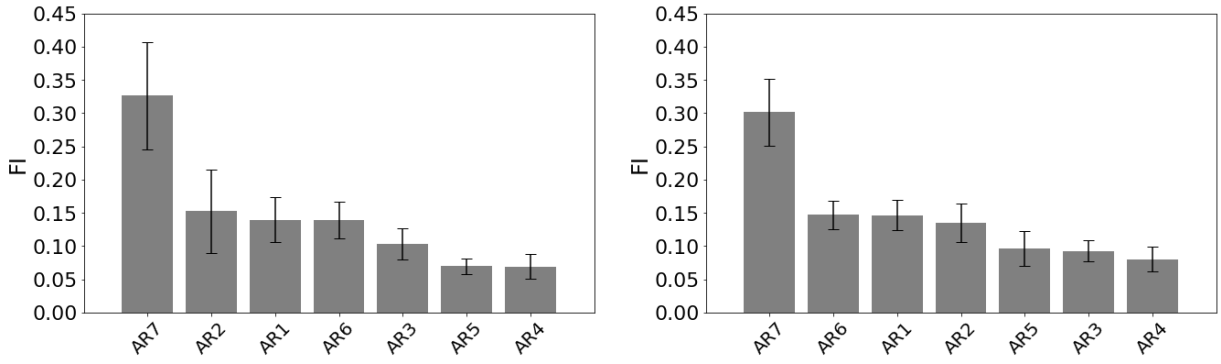
The confusion matrix analysis was conducted to check the possible existence of biased prediction, i.e., obtain a high accuracy by simply predicting one class (the majority) and missing the others (the minority) in an unbalanced dataset. For brevity, only the results of the ARI(7, 1) model were compared with the EVC in the two localization cases (damage state detection and pattern recognition) for floor 2, Fig. 14.12. High values were achieved on the diagonal cells while low values were on the off-diagonal ones, i.e., the ARIMA-ML algorithm predicted each class accurately without large bias. Comparing Figs 14.12a & 14.12b indicates that the algorithm was more accurate in distinguishing between undamaged and damaged states, but less accurate to identity the differences between damage levels (UD, PL, and AL). Moreover, Fig. 14.12b shows that the model was prone to predict "partial braces loosening" (PL) as being "all braced loosening" (AL) for the braces. This misclassification can be explained by a coupling effect from different floors. For example, there was only partial braces loosened in floors 1 & 2 but all braces were loosened in floor 3 after the MCE level of El Centro earthquake, Table 14.1, and the nonlinear response of floor 2 was influenced by that of floor 3 which had severe damage and also influenced by floor 1 which had similar damage level to floor 2. Thus, using the ARIMA model with such simple structural idealization (each floor treated independently) may not render very accurate damage pattern recognition task, even though the reported 85∼88% was still acceptable as a reasonably range of the accuracy, compared to the damage state detection task. The future work should make use of more accurate substructure idealization of the tested system [129].

The relatively low accuracy obtained by RF in both ARIMA(6, 1, 6) and ARIMA(7, 1, 6) compared to the other classifiers, Table 14.7, for the same features, indicates that the RF classifier may not work effectively with these features. Therefore, it is not reliable to rank the FI for these two models under these observed results and conditions. However, ARI(7, 1) produced comparable results of RF to the other classifies, where FI is shown in Figs 14.13a & 14.13b. It is observed that in ARI(7, 1), the importance of the high contribution terms decreased while that of the low contribution terms slightly increased. This is evident from



(a) Local damage state detection (b) Local damage pattern recognition

Figure 14.12: Confusion matrices of ARI(7, 1) with EVC for floor 2. (UD: undamaged state, D: damaged state, PL: partial braces loosening, & AL: all braces loosening)



(a) ARI(7, 1), floor 1 (b) ARI(7, 1), floor 2

Figure 14.13: FI of ARI(7, 1) model in local damage pattern recognition

the comparison with the FI results in the local damage state detection (Figs 14.9 & 14.13), where the FI of the AR order 7 decreased from 0.44 to 0.33 for floor 1 and 0.39 to 0.30 for floor 2, and FI values of the AR order 3, 4, or 5 increased by 0.03∼0.05. Therefore, this observation supports the significant contribution of high order AR coefficients, and shows the increasing contribution from lower order AR coefficients in the damage recognition tasks, especially going from local damage state detection to local damage pattern recognition.

14.6 Summary

In summary, to validate the effectiveness and robustness of the ARIMA-ML algorithm/framework on damage recognition of building structures, three detection experiments were considered: (1) general damage state detection, (2) local damage state detection, and (3) local damage pattern recognition. These were conducted based on data collected from shaking table tests for a steel structure with braces.

Along with different experiments, comparisons were made to explore the influence of overlapping while using the sliding window method in data segmentation and the effect of higher order terms in different candidate models, selected according to an effective EDA. The comparison with the experimental results demonstrated the robustness and accurate performance of the ARIMA-ML algorithm in all tasks, where 95%, 97%, and 88% accuracy were obtained for the above mentioned three tasks (1), (2), and (3), respectively. Finally, due to the nature of the RF classifier, FI was further analyzed to assess the feature performance. Specific conclusions are listed in the following:

- ARIMA coefficients are sensitive to the changes in stiffness (both global and local). Using these coefficients as features, the developed algorithm can accurately distinguish between the damaged and undamaged states with slightly lower performance in more fine-grained damage categories, i.e., multiple damage patterns.

- Overlapping pattern used in sliding window does not significantly affect the results, which is attributed to the size of the dataset (over 10,000 segments) generated by Non-OL. This dataset contains sufficient information to accurately train a classifier for global damage detection, and applying overlapping reduces the variance in the results but does not significantly improve the classification accuracy. It is noted that overlapping increases the computational cost and a trade-off between computational efficiency and results variance should be considered in different applications.

- EDA may lead to multiple satisfactory initial models and selecting several best models as candidate ones for damage detection makes the algorithm more robust. From experimental results, all candidate models perform similarly, but relatively simpler ones (e.g., ARI$(7, 1)$) can achieve better classification performance than more complex ones (e.g., ARIMA$(6, 1, 6)$ & ARIMA$(7, 1, 6)$) due to some overfitting in these more complex models. However, the ARIMA$(6, 1, 6)$ and ARIMA$(7, 1, 6)$ models are shown to have better fitting results through residual diagnosis, AIC, and BIC value evaluation. Therefore, it is recommended to consider several candidate models for damage identification instead of using one best fitted model.

- Ensemble EVC classifier shows more stable and accurate performance (at least as good as the best single ML classifier) with lower variance. Through a hard-voting mechanism, multiple individual ML algorithms are applied, and such mechanism improves the accuracy by avoiding overestimation and by reducing the variance of the results.

- Due to the good performance of RF, one important characteristic of RF, namely the FI, is evaluated to understand how each feature is weighted using this ML classifier for making the final decision. The results of FI indicate the influence of the higher order AR coefficients like AR 6 & 7 and some lower order MA coefficients like MA 1, 2 & 4. Therefore, it is important to engage all ARIMA coefficients as damage features for better and stable performance.

Even though ARIMA-ML achieved the promising results in the studied experiments, there still exist some drawbacks which need future studies. These are summarized as follows:

– In the current framework, only coefficient-based modeling is adopted in feature extraction, residual-based or even combining the two approaches is an important research direction to explore.

– Differencing raw acceleration data once, i.e., the *jerk* in control theory, lacks proper interpretation for damage identification through structural dynamics. This needs further theoretical development for justification.

– The ARIMA modeling and ML algorithms fit the available data with the possible risk of underfitting or overfitting, e.g., when using inappropriate orders in the ARIMA modeling or poorly tuned parameters in the ML algorithms. Even though selecting several candidate models instead of using one single model provides robustness of the algorithm, it is a subjective and costly approach, which requires further investigation.

– The presented algorithm was validated on shaking table tests of a TCBBF. More validations using physical tests or detailed finite element analyses are needed in the future. Moreover, using transfer learning (TL) techniques to only train on simulated data but fine-tune and test on real experimental data has the potential for more practical usage.

– Substructure idealization is suggested for the improvement in damage localization tasks of framed structures as the one studied in this chapter.

Chapter 15

Conclusions & Extensions

This dissertation introduced the fundamentals and applications of ML and DL in data-driven SHM via two directions, namely vision-based SHM and vibration-based SHM. Since the dissertation explored more aspects of the vision-based SHM, related topics in the vision-based SHM are further split into general and advance topics. Therefore, the dissertation consists of five core parts, namely:

1. Background & motivations of ML/DL-based data-driven SHM
2. General topics of vision-based SHM
3. Advanced topics of vision-based SHM
4. ML in vibration-based SHM
5. Conclusions & extensions

Background & motivations of ML/DL-based data-driven SHM

In the first part of the dissertation (Chapters 1 to 3), the motivations, background and the objectives of combining the ML and DL approaches with the civil and structural engineering applications, e.g., post-disaster structural reconnaissance and SHM, were briefly discussed. Then, theoretical foundation of ML and DL were covered to establish the bases for the subsequent parts of the dissertation.

General topics of vision-based SHM

In Chapter 4, the first large-scale and open-source image dataset with systematic framework, namely ϕ-Net, was proposed. This ϕ-Net, in its current status, defined eight benchmark detection tasks and relevant sub-datasets, namely: (1) scene level, (2) damage state, (3) spalling (material loss) condition, (4) material type, (5) collapse mode, (6) component type, (7) damage level, and (8) damage type. According to this framework, a large number of

structural images were collected, preprocessed, labeled, and split to form a large-scale dataset containing 36,413 pairs of images and labels. Furthermore, the dataset was separated into eight sub-datasets corresponding to the benchmark tasks and open-sourced online. These datasets are expected to benefit the relevant vision-based SHM studies in the future.

In Chapter 5, before conducting comprehensive numerical experiments on the eight sub-datasets of the ϕ-Net, a preliminary study was performed firstly on a smaller scale dataset for validating the effectiveness and applicability of DL methods in vision-based SHM using TL strategy with two patterns (feature extractor and fine-tuning). In the TL experiments, besides validating the effectiveness of the TL itself in alleviating over-fitting issues while training on small-scale dataset, the influence of network input size, retrain ratio, and different configurations of the deep CNN were also considered and investigated. The main conclusions from this part of the study are summarized as follows:

- TL is an effective training approach to mitigate the data deficiency in vision-based SHM tasks. Its feature extractor strategy acts as a preliminary computationally efficient analysis with a moderate recognition accuracy on the target dataset, and fine-tuning can be considered as an improvement tool beyond the feature extractor if good computational resources are available.

- The relatively optimal network input size for vision-based SHM is 224×224, and inadequate sizes would probably render worse results due to either lack of information or computational complexity and inefficiency.

- Some parameters in the last several Conv blocks represent structural damage features. In the retrain ratio experiment, VGG-type network with retraining two Conv blocks achieved quite promising accuracy for binary and three-class tasks and moderately acceptable results for the four-class damage type task.

- Both ResNet-50 and 101 have fast computational speed under feature extractor but with similar and limited accuracy especially for multi-class tasks. However, the performance of both ResNet-50 and 101 can be significantly improved through fine-tuning. Among the VGG-type, ResNet-50, and 101 models, the fine-tuned ResNet-101 is considered as the best model.

In Chapter 6, inspired and guided by the findings from the TL experiments, comprehensive benchmark experiments with respect to the ϕ-Net were conducted based on the eight benchmark datasets. Multiple factors were considered in a comparative study under a fair setting of hyper-parameters and training approaches, i.e., using DA and TL. Several key conclusions are summarized as follows:

- In a comparison of three DL models (VGG-16, VGG-19, and ResNet-50), ResNet-50 appeared to be more generalized with the highest test accuracy in most cases (except in Tasks 6 & 8 of the ϕ-Net).

- From the perspective of the training approach, TL worked well in all eight tasks, with significant improvement in many cases, but DA only had limited effects.
- As for computational efficiency, without DA, ResNet-50 cost the least computational time, VGG-16 cost the second least, and VGG-19 was the most expensive. With DA, all models had increasing time costs and similar training time per epoch due to the extra cost of performing affine transformations on the raw images. Moreover, while adopting TL, it did not affect the average training time, which indicates that TL is a more efficient and economical method for this type of the conducted experiments.
- For benchmarking purposes, reference results and history plots for all cases and tasks were presented, which provide the reference for future DL developments and applications related to the important field of vision-based SHM.

In Chapter 7, as direct applications of ϕ-Net, two examples were introduced. The first one is an image-based post-disaster damage assessment of the 1999 Chi-Chi earthquake, where the best DL models trained from the benchmark experiments were directly applied to extract attributes from images collected after the earthquake. An acceptable performance in both accuracy and run-time was captured, which can further provide information for decision-making purposes. The second one is the 2018 ϕ-Net Challenge held prior to open-sourcing the ϕ-Net dataset. It is the first image-based challenge in structural engineering, which encouraged transdisciplinary collaboration.

In this part of the dissertation, from Chapters 4 to 7, the discussed DL technologies brought new directions for the detection and evaluation of structural damage from images, showing a satisfying performance and great potential in civil and structural engineering applications. However, these studies are still preliminary, and there are several extensions to be explored as follows:

- The ϕ-Net framework is generic, where only eight tasks were explored in this dissertation, and it can be further expanded with more fine-grained categories to better represent the complexity of the structural systems in the built environment. In addition, more general CV problems other than classification, e.g., object detection, localization, and semantic segmentation can be explored.
- Compared to the benchmark datasets used in the CV domain, the scale of the ϕ-Net is still small. However, it is expected to enlarge the dataset with more labeled data. It is recommended to make full use of the ϕ-Net dataset to crowd-source the users so they contribute their own datasets, to quantify finer categories, and to relabel images for new vision tasks, among several other possible extensions.
- In order to obtain better results, tuning hyper-parameters (e.g., learning rate, dropout rate, and the number of neurons in the fc-layers), applying other deep CNN models, e.g., InceptionNet [122], and developing new models and training approaches are all proposed for future studies.

– Once the DL model with stable performance is achieved, it can be programmed into portable hardware or as mobile software application (app) combined with prevailing drone/robot-based detection systems [85, 142] to help with real-time structural damage recognition and evaluation for effective decision-making in post-disaster reconnaissance efforts and SHM.

Even though promising results were achieved in DL-based methods in vision-based SHM problems, their internal working principle are still hard to understand by human experts, and thus such ML methods are treated as a "black box". To address this issue, Chapters 8 & 9 discussed the latest visual interpretation methods in explaining the DL model in SHM problems. In Chapter 8, several basics and background information about DL visual interpretation methods were introduced first. Then, in Chapter 9, numerical experiments were conducted to apply these novel methods in interpreting how DL models work in structural vision tasks. The applications of recent widely-used visual interpretable methods, namely GBP, CAM, Grad-CAM, and Guided Grad-CAM, were investigated and applied to the basic SHM vision tasks. Beyond the general view over the basic (non-expert) spalling condition task, several key factors including learning procedure, learned features from different network depths, TL, DA, and the level of semantic abstraction were explored. Through comprehensive experiments, several key conclusions are summarized as follows:

- Grad-CAM and Guided Grad-CAM performed well in interpreting a well-trained deep CNN model for the spalling conditions. The visual interpretable qualities are related to the scene level of the images, model recognition ability and preference, where the best results were obtained in the pixel level SP images.

- While identifying spalling, ellipse-like regions are usually used as model evidence. The model is also capable of finding some tiny spalled areas. However, its performance degraded with the increase of image complexity and some other small or uncommon damage patterns.

- VGG-type model using TL usually learned rapidly in the first several epochs. The active regions in the heatmap of Grad-CAM and Guided Grad-CAM changed to accommodate the increasing discriminative ability of the model, which may sometimes finally end up with an incomplete coverage of the damaged area. The model is always sensitive to capturing line-like edge features from the GBP results

- From the saliency maps of the GBP, the model started to have some sense of the damage features from shallow and intermediate layers of the CNN, and the discriminative ability was achieved in the deep layers of the CNN.

- The noisy and unreasonable heatmap results of the baseline model initialized from a random parameter space indicate that it is incapable of extracting damage features and lacks of discriminative ability, which can be significantly alleviated by using TL.

- Worse visual results were obtained in more abstract damage level task, e.g., the damage type task. With the limited training data, it is difficult for the current DL model to recognize well the abstract semantic meaning of the damage type, especially for flexural damage. However, some saliency maps of Grad-CAM and Guide Grad-CAM show that the model still has the sense of recognizing specific damage patterns with respect to shear and combined damage types.

- From the perspective of model performance, visual explanation provided by the saliency maps gave more intuitions to improve the model, e.g., by enlarging the training dataset, further splitting the data into more fine-grained scene levels and subsequently conducting classification within each level.

As an early study in the direction of interpretation of the vision-based SHM using DL, there are several extensions which are summarized as follows:

– The issue of incomplete active region coverage may be related to the limitation of the Grad-CAM method. Thus, several of the state-of-the-art methods, e.g., Grad-CAM++ [17], are expected to be explored for better visualization in the future.

– For comprehensive experiments, only GBP, Grad-CAM, and Guided Grad-CAM were compared, and other interpretation methods, e.g., LIME [99], need further discussions.

– More applications and extensions of the visual interpretation approaches need further investigations, e.g., model selection based on the understanding obtained by visual methods has great potentials and benefits in practice.

Advanced topics of vision-based SHM

In Chapters 10 & 11, as an alternative method to relieve data deficit issue and to further augment the limited data resources using methods other than the conventional DA, GAN was introduced for the first time in the vision-based SHM area. As the new study in this area, two major detection tasks (scene level identification and damage state check) were mainly studied based on a simplified ϕ-Net representations (depth-2 and depth-3). Due to the complex distribution of the structural images, common schema of GAN training does not work well and only results into some mixed synthetic images. Thus, a well-designed DCGAN architecture and Leaf-Bootstrapping (LB) algorithm aiming towards learning the data distribution from a combination of its sub-classes were proposed to improve the synthetic image quality. In addition to the usual human inspection for the evaluation of the GAN performance, Self-Inception Score (SIS) and Generalization Ability (GA) were proposed to complement the human inspection for the evaluation of the GAN performance. Moreover, four experimental cases were designed to explore the performance of LB-based GAN under different situations with the above-mentioned metrics. In addition, to further explore the potentials of using synthetic images to improve the classifier performance, a simple and

well-designed experiment, namely synthetic data augmentation, in a classification task was conducted under the constraints of limited computational resources and low-data regime. In that regard, a union training strategy, namely synthetic data fine-tuning (SDF), was proposed and applied. Some conclusions from these experiments are summarized as follows:

- While adopting shallow representation of the ϕ-Net framework, DCGAN achieved a stable and satisfactory performance in generating reasonable structural images in all levels, which outperforms directly training schema of GAN.
- Further data cleaning helps generate clearer images, but increases training difficulties and the risks of mode collapse issues. Conventional DA and using deeper representations also help with improving the visual qualities in most cases.
- SIS and GA indicate similar trends in the evaluation procedure to those of the human-expert judgement under the designed experimental settings.
- Directly training a weak classifier with a mixed dataset by real and synthetic images under the low-data regime and limited computational resources did not work and even rendered worse results than a baseline model. However, the proposed SDF pipeline can enhance the performance, and achieve the best results compared to other traditional training pipelines.

Summarizing, the DCGAN with the LB algorithm can achieve the state-of-the-art performance in generating synthetic images of structures with both damaged and undamaged states, and the SDF pipeline indicates the great potentials of GAN in improve model performance in classification problems. However, as the first exploration in this area, several activities can be listed as future extensions as follows:

- There are over 300 kinds of GANs designed. The performance of other kinds of GAN models is worth exploring in the future research.
- Besides SIS and GA, there are many metrics to be explored in the future, e.g., Wasserstein Distance for applying WGAN, Fréchet Inception Distance (FID) proposed by [44].
- Even though the SDF pipeline outperformed other traditional training pipelines in the synthetic data augmentation, it is still time-consuming and inefficient with generated data compared to conventional DA, and more explorations are expected in the future by using the techniques of semi-supervised learning.
- More data aggregation applications, other than classification, need to be explored, e.g., damage localization, semantic segmentation, etc.
- More future studies should be considered to encode the inclusion of the laws of physics and logical information into the GAN model formulation to produce more convincing practical models.

In Chapter 12, to further address the inefficiency issues reported in the GAN experiments, a first-of-its-kind study converting the GAN into a semi-supervised learning approach was developed and investigated. Furthermore, besides considering the limitations of the low-data regime and limited computational power, imbalanced-class(label) issues was also considered through experiments. Therefore, a semi-supervised learning GAN pipeline along with the balanced batch sampling technique was proposed and named BSS-GAN. A series of computer experiments for the cover spalling detection in reinforced concrete structures was designed and conducted, where comparisons and discussions were made between a baseline (conventional training pipeline), referred to as BSL, BSS-GAN, and synthetic data augmentation (GAN-Aug). several key observations and conclusion are listed as follows:

- Both baseline and BSS-GAN share similar trends in the overall accuracy and the true positive rate (TPR), where the increase of labeled ratio led to the general increase of the accuracy and the TPR. Moreover, BSS-GAN outperformed BSL for all cases with different amounts of labeled data.

- Using 100% labeled data, BSS-GAN still achieved the best overall accuracy and TPR than the BSL and the GAN-Aug, which makes the BSS-GAN the "state-of-the-art" GAN-related pipeline considering the conducted limited experiments.

- As for synthetic image quality, there is no evident model collapse issues occurred in the results of the BSS-GAN, and the synthetic images have good visual variety. However, the generated images of the BSS-GAN are blurry and less realistic than those produced by the generator of the ordinary GAN. This is mainly due to (a) the loss function of BSS-GAN focusing more on the classification task instead of purely conducting image generation, and (b) the lack of conditioning operations in the BSS-GAN, e.g., separating and filtering data for complexity reduction. Unlike the LB-GAN (presented in Chapter 11), which is focused on training images for a specific class label having a relatively simple distribution, the generator in the BSS-GAN receives all images, which force the generator to learn complex and mixed distributions consisting of multiple classes. Thus, it leads to the inferior generating performance of the BSS-GAN compared to the original GAN using the LB method.

As a preliminary study, there still exist several topics that need to be explored in the future. These are listed as follows:

– The BSS-GAN was only shown to be effective under the low-data regime, more experiments need to be conducted for medium-data regime or even large datasets, which can be realized by continuing the efforts in collecting and labeling structural images in practice to expand the existing ϕ-Net dataset.

– only one CNN architecture was tested herein, and more parametric studies with respect to other network architectures are expected in the future.

ML in vibration-based SHM

In the fourth part of the dissertation (Chapters 13 & 14), the scope of using ML/DL in SHM was extended to the vibration-based SHM, making a logical extension towards the data-driven vibration-based SHM. TS analysis is one of the most common and classic methods in analyzing vibration data, which is very relevant to SHM of structures subjected to dynamic loading or temperature changes. The structural responses like acceleration, velocity, and displacement collected from sensors deployed on the structural systems (e.g., buildings and bridges) directly belong to the scope of TS. Therefore, both TS modeling and ML classification were combined together to solve one kind of important and widely encountered problem in the vibration-based SHM applications, namely damage recognition in building structures.

While discussing the TS modeling for the analysis of the structural responses, some limitations, e.g., non-stationarity and subjective feature selection, reduced the wide-spread use of TS modeling in practice. Thus, to overcome the constraints of the stationary condition requirements in TS modeling, ARIMA model was thought of as a candidate for fitting more general structural TS signals. Furthermore, in order to combine TS with ML classification together, ARIMA-ML algorithm/framework was developed, which contains the following modules: data preprocessing, stationarity check, feature extraction, and ML classification. Shaking table experimental data from a tension-only concentrically braced beam-through frame (TCBBF) made of steel together with findings from previous studies [87, 90], were utilized for algorithm validation. Three different detection tasks were tested, namely (1) general damage state detection, (2) local damage state detection, and (3) local damage pattern recognition. Besides these different tasks, comparisons were also made to explore the influence of: (1) overlapping while using the sliding window method in data segmentation, and (2) higher order terms in different candidate models selected according to an exploratory data analysis (EDA). Experimental results indicate the robustness and promising performance of the ARIMA-ML algorithm in all considered tasks, and several key conclusions are summarized as follows:

- Overlapping does not have significant influence on the results. It helps with reducing variance of the results but may not greatly improve the classification performance. Applying Overlapping also increases the computational time. Thus, there is a trade-off between computational efficiency and results variance for practical usage.

- EDA may lead to multiple satisfactory models, and selecting several best models as candidate models for damage detection will make the algorithm more robust.

- Ensemble voting classifier (EVC) for the considered four ML algoithms achieved a stable and accurate performance which is at least as good as the best individual ML algorithm with lower variance. Through a hard voting mechanism, multiple individual ML algorithms can help the EVC to avoid overestimating and reduce the variance of the results.

- The feature importance (FI) analysis helps to understand how the RF classifier, weights each feature for the final decision. The results of FI also indicate the influence of higher order AR coefficients like AR 6 & 7 and some lower order MA coefficients like MA 1, 2 & 4, which validated the necessity to engage all coefficients of the ARIMA model as damage features for better and stable classification performance.

Even though the ARIMA-ML model achieved promising results in the designed experiments, several limitations and future extensions are suggested as summarized below:

- Both procedures of using ARIMA modeling and ML algorithms need to fit the data, which increases the risk for under- or over-fitting if either procedure is operated improperly. This observation requires further investigation both theoretically and numerically considering several practical applications.
- Differencing operation on acceleration data lacks of physical interpretation, which needs further theoretical study.
- Candidate model selection is still subjective and more quantitative metrics are expected to be proposed in the future.
- More experiments based on various data sources, e.g., different shaking table tests or responses from sensors deployed on real structures, are expected in the future. The concept of TL that only train the ML model on simulated data but test on the real experimental data is also worthy of in-depth future investigations to render ML/DL technologies more appealing to the broader applications related to SHM of the built environment.

A closure

This dissertation presented a comprehensive treatise about the foundation and applications of machine and deep learning methods in the context of the data-driven structural health monitoring. The developed advances and shown promising results achieved herein shed light on the great potentials of these state-of-the-art methodologies in relevant future study and practical structural engineering applications. In these future pursuits, improved monitoring, learning, maintenance, and ultimately effective decision-making regarding the conditions, replacement or retrofits of the built environment can be reliably achieved.

Bibliography

[1] HOJJAT ADELI and OSAMA KAMAL. "Parallel structural analysis using threads". In: *Computer-Aided Civil and Infrastructure Engineering* 4.2 (1989), pp. 133–147.

[2] Hojjat Adeli and Osama Kamal. *Parallel processing in structural engineering*. CRC Press, 1993.

[3] Rabih Amhaz et al. "Automatic crack detection on two-dimensional pavement images: An algorithm based on minimal path selection". In: *IEEE Transactions on Intelligent Transportation Systems* 17.10 (2016), pp. 2718–2729.

[4] Antreas Antoniou, Amos Storkey, and Harrison Edwards. "Data augmentation generative adversarial networks". In: *arXiv preprint arXiv:1711.04340* (2017).

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[6] Shane Barratt and Rishi Sharma. "A note on the inception score". In: *arXiv preprint arXiv:1801.01973* (2018).

[7] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 17–36.

[8] David Berthelot, Thomas Schumm, and Luke Metz. "Began: Boundary equilibrium generative adversarial networks". In: *arXiv preprint arXiv:1703.10717* (2017).

[9] Christopher M Bishop. *Pattern recognition and machine learning*. New York: springer, 2006.

[10] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[11] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[12] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018).

[13] Lorenzo Bruzzone, Roberto Cossu, and Gianni Vernazza. "Combining parametric and non-parametric algorithms for a partially unsupervised classification of multitemporal remote-sensing images". In: *Information Fusion* 3.4 (2002), pp. 289–297.

[14] Young-Jin Cha, Wooram Choi, and Oral Büyüköztürk. "Deep learning-based crack damage detection using convolutional neural networks". In: *Computer-Aided Civil and Infrastructure Engineering* 32.5 (2017), pp. 361–378.

[15] Young-Jin Cha et al. "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types". In: *Computer-Aided Civil and Infrastructure Engineering* 33.9 (2018), pp. 731–747.

[16] Nicholas Charron et al. "Automated Bridge Inspection Using Mobile Ground Robotics". In: *Journal of Structural Engineering* 145.11 (2019), p. 04019137.

[17] Aditya Chattopadhay et al. "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 839–847.

[18] Yueshi Chen, Wei Wang, and Yiyi Chen. "Full-scale shake table tests of the tension-only concentrically braced steel beam-through frame". In: *Journal of Constructional Steel Research* 148 (2018), pp. 611–626.

[19] A.K. Chopra. *Dynamics of Structures: Theory and Applications to Earthquake Engineering*. Civil Engineering and Engineering Mechanics Series. Prentice Hall, 2012. ISBN: 9780132858038. URL: `https://books.google.com/books?id=3cctkgEACAAJ`.

[20] China Ministry of Construction (CMC). *Code for Seismic Design of Buildings (GB50011-2010)*. 2010.

[21] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[22] Jonathan D Cryer and Natalie Kellet. *Time series analysis*. Springer, 1991.

[23] Oliver Richard De Lautour and Piotr Omenzetter. "Detection of seismic damage in buildings using time series analysis and pattern recognition". In: *Proceedings of the 4th World Conference on Structural Control and Monitoring*. 2006, pp. 1–8.

[24] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition* (2009), pp. 248–255.

[25] Laigen Dong and Jie Shan. "A comprehensive review of earthquake-induced building damage detection with remote sensing techniques". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 84 (2013), pp. 85–99.

[26] Sattar Dorafshan, Robert J Thomas, and Marc Maguire. "Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete". In: *Construction and Building Materials* 186 (2018), pp. 1031–1045.

[27] Ding-Zhu Du and Panos M Pardalos. *Minimax and applications*. Vol. 4. Springer Science & Business Media, 2013.

[28] PJ Fanning and EP Carden. "Auto-regression and statistical process control techniques applied to damage indication in telecommunication masts". In: *Key Engineering Materials*. Vol. 204. Trans Tech Publ. 2001, pp. 251–260.

[29] Reza V Farahani and Dayakar Penumadu. "Full-scale bridge damage identification using time series analysis of a dense array of geophones excited by drop weight". In: *Structural Control and Health Monitoring* 23.7 (2016), pp. 982–997.

[30] Charles R Farrar et al. "A statistical pattern recognition paradigm for vibration-based structural health monitoring". In: *Structural Health Monitoring* 2000 (1999), pp. 764–773.

[31] Dongming Feng and Maria Q Feng. "Experimental validation of cost-effective vision-based structural health monitoring". In: *Mechanical Systems and Signal Processing* 88 (2017), pp. 199–211.

[32] Maayan Frid-Adar et al. "Synthetic data augmentation using GAN for improved liver lesion classification". In: *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*. IEEE. 2018, pp. 289–293.

[33] Michael L Fugate, Hoon Sohn, and Charles R Farrar. "Vibration-based damage detection using statistical process control". In: *Mechanical Systems and Signal Processing* 15.4 (2001), pp. 707–721.

[34] Wayne A Fuller. *Introduction to statistical time series*. Vol. 428. John Wiley & Sons, 2009.

[35] Corrado Gini. "Variabilità e mutabilità". In: *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi* (1912).

[36] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[37] Ian Goodfellow. "NIPS 2016 tutorial: Generative adversarial networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge: MIT Press, 2016.

[39] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[40] Baptiste Gregorutti, Bertrand Michel, and Philippe Saint-Pierre. "Correlation and variable importance in random forests". In: *Statistics and Computing* 27.3 (2017), pp. 659–678.

[41] James D Hamilton. *Time series analysis*. Vol. 2. Princeton New Jersey, 1994.

[42] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[43] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[44] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in neural information processing systems*. 2017, pp. 6626–6637.

[45] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[46] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[47] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[48] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[49] Jochen Jaeger and Martin Scheringer. "Transdisciplinary research: Why is it still so rare, and why should we care?" In: *Balance-Unbalance International Conference,* `http://balance-unbalance2017.org/` (2017).

[50] Yanghua Jin et al. "Towards the automatic anime characters creation with generative adversarial networks". In: *arXiv preprint arXiv:1708.05509* (2017).

[51] Tung Khuc and F Necati Catbas. "Structural identification using computer vision–based bridge health monitoring". In: *Journal of Structural Engineering* 144.2 (2018), p. 04017202.

[52] Jinkyu Kim and John Canny. "Interpretable learning for self-driving cars by visualizing causal attention". In: *Proceedings of the IEEE international conference on computer vision*. 2018, pp. 2942–2950.

[53] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[54] Andy Kitchen and Jarrel Seah. "Deep generative adversarial neural networks for realistic prostate lesion MRI synthesis". In: *arXiv preprint arXiv:1708.00129* (2017).

[55] Christian Koch et al. "A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure". In: *Advanced Engineering Informatics* 29.2 (2015), pp. 196–210.

[56] R Kohavi and F Provost. "Confusion matrix". In: *Machine learning* 30.2-3 (1998), pp. 271–274.

[57] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *Technical Report TR-2009* (2009).

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[59] K Lakshmi, A Rama Mohan Rao, and N Gopalakrishnan. "Singular spectrum analysis combined with ARMAX model for structural damage detection". In: *Structural Control and Health Monitoring* 24.9 (2017), e1960.

[60] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[61] Yann LeCun et al. "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[62] Christian Ledig et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[63] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[64] Y Lei et al. "Statistical damage detection using time series analysis on a structural health monitoring benchmark problem". In: *Proceedings of the 9th international conference on applications of statistics and probability in civil engineering*. 2003, pp. 6–9.

[65] Bixiong Li and Khalid M Mosalam. "Seismic performance of reinforced-concrete stairways during the 2008 Wenchuan earthquake". In: *Journal of performance of constructed facilities* 27.6 (2013), pp. 721–730.

[66] Hang Li. *Statistical learning methods (in Chinese)*. Tsinghua University Press, 2012.

[67] Kunpeng Li et al. "Tell me where to look: Guided attention inference network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9215–9223.

[68] Xiao Liang. "Image-based post-disaster inspection of reinforced concrete bridge systems using deep learning with Bayesian optimization". In: *Computer-Aided Civil and Infrastructure Engineering* 34.5 (2019), pp. 415–430.

[69] Heng Liu and Yunfeng Zhang. "Image-driven structural steel damage condition assessment method using deep learning algorithm". In: *Measurement* 133 (2019), pp. 168–181.

[70] Greta M Ljung and George EP Box. "On a measure of lack of fit in time series models". In: *Biometrika* 65.2 (1978), pp. 297–303.

[71] Jerome Peter Lynch. "Linear classification of system poles for structural damage detection using piezoelectric active sensors". In: *Smart Structures and Materials 2004: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*. Vol. 5391. International Society for Optics and Photonics. 2004, pp. 9–20.

[72] Jerome Peter Lynch et al. "Embedment of structural monitoring algorithms in a wireless sensing unit". In: *Structural Engineering and Mechanics* 15.3 (2003), pp. 285–297.

[73] Yichi Ma and Muhan Ma. "A novel approach to artistic textual visualization via GAN". In: *Proceedings of the 2017 International Conference on Information Technology*. 2017, pp. 130–135.

[74] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[75] Ali Madani et al. "Chest x-ray generation and data augmentation for cardiovascular abnormality classification". In: *Medical Imaging 2018: Image Processing*. Vol. 10574. International Society for Optics and Photonics. 2018, p. 105741M.

[76] Ali Madani et al. "Semi-supervised learning with generative adversarial networks for chest x-ray classification with ability of data domain adaptation". In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE. 2018, pp. 1038–1042.

[77] Marc Maguire, Sattar Dorafshan, and Robert J Thomas. "SDNET2018: A concrete crack image dataset for machine learning applications". In: (2018).

[78] Giovanni Mariani et al. "Bagan: Data augmentation with balancing gan". In: *arXiv preprint arXiv:1803.09655* (2018).

[79] Steven G Mattson and Sudhakar M Pandit. "Statistical moments of autoregressive model residuals for damage localisation". In: *Mechanical Systems and Signal Processing* 20.3 (2006), pp. 627–645.

[80] Liu Mei, Akira Mita, and Jin Zhou. "An improved substructural damage detection approach of shear structure based on ARMAX model residual". In: *Structural Control and Health Monitoring* 23.2 (2016), pp. 218–236.

[81] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[82] Jack Moehle. *Seismic design of reinforced concrete buildings*. McGraw Hill Professional, 2014.

[83] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. Cambridge: MIT press, 2012.

[84] Christopher F Mooney et al. *Bootstrapping: A nonparametric approach to statistical inference*. 95. sage, 1993.

[85] G Morgenthal and N Hallermann. "Quality assessment of unmanned aerial vehicle (UAV) based visual inspection of structures". In: *Advances in Structural Engineering* 17.3 (2014), pp. 289–302.

[86] Jawad Nagi et al. "Max-pooling convolutional neural networks for vision-based hand gesture recognition". In: *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE. 2011, pp. 342–347.

[87] K Krishnan Nair, Anne S Kiremidjian, and Kincho H Law. "Time series-based damage detection and localization algorithm with application to the ASCE benchmark structure". In: *Journal of Sound and Vibration* 291.1-2 (2006), pp. 349–368.

[88] KK Nair et al. "Application of time series analysis in structural damage evaluation". In: *Proceedings of the international conference on structural health monitoring*. 2003, pp. 14–16.

[89] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[90] Hae Young Noh et al. "Application of time series based damage detection algorithms to the benchmark experiment at the National Center for Research on Earthquake Engineering (NCREE) in Taipei, Taiwan". In: *Smart Structures and Systems* 5.1 (2009), pp. 95–117.

[91] Piotr Omenzetter and James Mark William Brownjohn. "Application of time series analysis for bridge monitoring". In: *Smart Materials and Structures* 15.1 (2006), p. 129.

[92] Maxime Oquab et al. "Learning and transferring mid-level image representations using convolutional neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[93] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[94] Luis Perez and Jason Wang. "The effectiveness of data augmentation in image classification using deep learning". In: *arXiv preprint arXiv:1712.04621* (2017).

[95] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[96] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[97] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[98] Muhammad Imran Razzak, Saeeda Naz, and Ahmad Zaib. *Classification in BioApps*. New York: Springer, 2018.

[99] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Model-agnostic interpretability of machine learning". In: *arXiv preprint arXiv:1606.05386* (2016).

[100] Richard J Rossi. *Mathematical statistics: an introduction to likelihood based inference*. John Wiley & Sons, 2018.

[101] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[102] Tim Salimans et al. "Improved techniques for training gans". In: *Advances in neural information processing systems*. 2016, pp. 2234–2242.

[103] João Pedro Santos et al. "Early damage detection based on pattern recognition and data fusion". In: *Journal of Structural Engineering* 143.2 (2017), p. 04016162.

[104] Mohamed A El-Sayed, Yarub A Estaitia, and Mohamed A Khafagy. "Automated edge detection using convolutional neural network". In: *Int. J. Adv. Comput. Sci. Appl.(IJACSA)* 4.10 (2013).

[105] Ramprasaath R Selvaraju et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[106] H Sezen et al. "Performance of reinforced concrete buildings during the August 17, 1999 Kocaeli, Turkey earthquake, and seismic design and construction practise in Turkey". In: *Engineering Structures* 25.1 (2003), pp. 103–114.

[107] Ling Shao, Fan Zhu, and Xuelong Li. "Transfer learning for visual categorization: A survey". In: *IEEE transactions on neural networks and learning systems* 26.5 (2014), pp. 1019–1034.

[108] Yong Shi et al. "Automatic road crack detection using random structured forests". In: *IEEE Transactions on Intelligent Transportation Systems* 17.12 (2016), pp. 3434–3445.

[109] Hoo-Chang Shin et al. "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning". In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1285–1298.

[110] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3145–3153.

[111] Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples*. Springer, 2011.

[112] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).

[113] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[114] Hoon Sohn and Charles R Farrar. "Damage diagnosis using time series analysis of vibration signals". In: *Smart materials and structures* 10.3 (2001), p. 446.

[115] Hoon Sohn et al. *Applying the LANL statistical pattern recognition paradigm for structural health monitoring to data from a surface-effect fast patrol boat*. Tech. rep. Los Alamos National Lab, 2001.

[116] Yan Song, Ian McLoughLin, and Lirong Dai. "Deep bottleneck feature for image classification". In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. 2015, pp. 491–494.

[117] Daniel Soukup and Reinhold Huber-Mörk. "Convolutional neural networks for steel surface defect detection from photometric stereo images". In: *International Symposium on Visual Computing*. Springer. 2014, pp. 668–677.

[118] Jost Tobias Springenberg. "Unsupervised and semi-supervised learning with categorical generative adversarial networks". In: *arXiv preprint arXiv:1511.06390* (2015).

[119] Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806* (2014).

[120] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[121] Frantisek Stulajter. *Predictions in time series using regression models*. Springer Science & Business Media, 2013.

[122] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[123] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[124] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[125] Matthew M Torok, Mani Golparvar-Fard, and Kevin B Kochersberger. "Image-based automated 3D crack detection for post-disaster building assessment". In: *Journal of Computing in Civil Engineering* 28.5 (2014), A4014004.

[126] Andrea Vedaldi and Stefano Soatto. "Quick shift and kernel methods for mode seeking". In: *European conference on computer vision*. Springer. 2008, pp. 705–718.

[127] Anand Vetrivel et al. "Disaster damage detection through synergistic use of deep learning and 3D point cloud features derived from very high resolution oblique aerial images, and multiple-kernel-learning". In: *ISPRS journal of photogrammetry and remote sensing* 140 (2018), pp. 45–59.

[128] Saining Xie et al. "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[129] Zhenhua Xing and Akira Mita. "A substructure approach to local damage detection of shear structure". In: *Structural Control and Health Monitoring* 19.2 (2012), pp. 309–318.

[130] Yang Xu et al. "Identification framework for cracks on a steel structure surface by a restricted Boltzmann machines algorithm based on consumer-grade camera images". In: *Structural Control and Health Monitoring* 25.2 (2018), pp. 1031–1045.

[131] Yang Xu et al. "Surface fatigue crack identification in steel box girder of bridges by a deep fusion convolutional neural network based on consumer-grade camera images". In: *Structural Health Monitoring* 18.3 (2019), pp. 653–674.

[132] Yadong Xue and Yicheng Li. "A fast detection method via region-based fully convolutional neural networks for shield tunnel lining defects". In: *Computer-Aided Civil and Infrastructure Engineering* 33.8 (2018), pp. 638–654.

[133] Na Yang and Xiaobin Bai. "Forecasting structural strains from long-term monitoring data of a traditional T ibetan building". In: *Structural Control and Health Monitoring* 26.1 (2019), e2300.

[134] Yongchao Yang and Satish Nagarajaiah. "Dynamic imaging: real-time detection of local structural damage with blind separation of low-rank background and sparse innovation". In: *Journal of Structural Engineering* 142.2 (2016), p. 04015144.

[135] Chul Min Yeum and Shirley J Dyke. "Vision-based automated crack detection for bridge inspection". In: *Computer-Aided Civil and Infrastructure Engineering* 30.10 (2015), pp. 759–770.

[136] Chul Min Yeum, Shirley J Dyke, and Julio Ramirez. "Visual data classification in post-event building reconnaissance". In: *Engineering Structures* 155 (2018), pp. 16–24.

[137] Chul Min Yeum et al. "Big visual data analytics for damage classification in civil engineering". In: *Transforming the Future of Infrastructure through Smarter Information: Proceedings of the International Conference on Smart Infrastructure and Construction, 27–29 June 2016*. ICE Publishing, London, UK. 2016, pp. 569–574.

[138] Hyungchul Yoon et al. "Target-free approach for vision-based structural system identification using consumer-grade cameras". In: *Structural Control and Health Monitoring* 23.12 (2016), pp. 1405–1416.

[139] Fisher Yu et al. "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop". In: *arXiv preprint arXiv:1506.03365* (2015).

[140] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[141] Allen Zhang et al. "Automated pixel-level pavement crack detection on 3D asphalt surfaces using a deep-learning network". In: *Computer-Aided Civil and Infrastructure Engineering* 32.10 (2017), pp. 805–819.

[142] Chunsun Zhang and Ahmed Elaksher. "An Unmanned Aerial Vehicle-Based Imaging System for 3D Measurement of Unpaved Road Surface Distresses 1". In: *Computer-Aided Civil and Infrastructure Engineering* 27.2 (2012), pp. 118–129.

[143] Wendong Zhang et al. "Depth structure preserving scene image generation". In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 727–736.

[144] Xiaofeng Zhang et al. "Dada: Deep adversarial data augmentation for extremely low data regime classification". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 2807–2811.

[145] Zhedong Zheng, Liang Zheng, and Yi Yang. "Unlabeled samples generated by gan improve the person re-identification baseline in vitro". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3754–3762.

[146] Bolei Zhou et al. "Learning deep features for discriminative localization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.