UNIVERSITY OF CALIFORNIA SAN DIEGO

Towards Universal Object Detection

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Signal and Image Processing)

by

Zhaowei Cai

Committee in charge:

Professor Nuno Vasconcelos, Chair
Professor Kenneth Kreutz-Delgado
Professor David Kriegman
Professor Truong Nguyen
Professor Zhuowen Tu

2019

The dissertation of Zhaowei Cai is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2019

DEDICATION

To my family.

EPIGRAPH

Look deep into nature, and then you will understand everything better.

—Albert Einstein

TABLE OF CONTENTS

Signature Page . . . iii

Dedication . . . iv

Epigraph . . . v

Table of Contents . . . vi

List of Figures . . . x

List of Tables . . . xii

Acknowledgements . . . xiv

Vita . . . xvii

Abstract of the Dissertation . . . xviii

Chapter 1 Introduction . . . 1
1.1 Object Detection . . . 3
1.2 Challenges on Object Detection Universality . . . 3
1.3 Contributions of the Thesis . . . 6
1.3.1 An Efficient Framework for Multi-Scale Detection . . . 6
1.3.2 An Effective Architecture for High-Quality Detection . . . 7
1.3.3 A Novel Design for Domain-Universal Detection . . . 8
1.3.4 Optimal Detector Solutions under Complexity Constraint . . . 8
1.4 Organization of the Thesis . . . 10

Chapter 2 Multi-Scale Object Detection . . . 11
2.1 Introduction . . . 12
2.2 Related Work . . . 14
2.3 Multi-Scale Object Proposal Network . . . 15
2.3.1 Multi-Scale Detection Strategies . . . 16
2.3.2 Architecture . . . 17
2.3.3 Sampling . . . 19
2.3.4 Implementation Details . . . 20
2.4 Object Detection Network . . . 21
2.4.1 CNN Feature Map Approximation . . . 22
2.4.2 Context Embedding . . . 23
2.4.3 Implementation Details . . . 23
2.5 Experimental Evaluation . . . 24
2.5.1 Proposal Evaluation . . . 25

2.5.2 Object Detection Evaluation 27
2.6 Conclusions 30
2.7 Acknowledgements 30

Chapter 3 High-Quality Object Detection 31
3.1 Introduction 32
3.2 Related Work 37
3.3 Review of High-Quality Object Detection 40
3.3.1 Object Detection 40
3.3.2 Detection Quality 42
3.3.3 Challenges to High Quality Detection 43
3.4 Cascade R-CNN 45
3.4.1 Architecture 45
3.4.2 Cascaded Bounding Box Regression 45
3.4.3 Cascaded Detection 46
3.4.4 Differences from Previous Works 48
3.5 Instance Segmentation 50
3.5.1 Mask R-CNN 50
3.5.2 Cascade Mask R-CNN 50
3.6 Experimental Results 52
3.6.1 Experimental Set-up 52
3.6.2 Quality Mismatch 54
3.6.3 Comparison with *Iterative BBox* and *Integral Loss* 56
3.6.4 Ablation Experiments 58
3.6.5 Comparison with the state-of-the-art 60
3.6.6 Generalization Capacity 63
3.6.7 Proposal Evaluation 65
3.6.8 Instance Segmentation by Cascade Mask R-CNN 65
3.6.9 Results on PASCAL VOC 66
3.6.10 Additional Results on other Datasets 68
3.7 Conclusion 70
3.8 Acknowledgements 71

Chapter 4 Domain-Universal Object Detection 72
4.1 Introduction 73
4.2 Related Work 76
4.3 Multi-Domain Object Detection 78
4.3.1 Universal Object Detection Benchmark 78
4.3.2 Single-Domain Detector Bank 79
4.3.3 Adaptive Multi-Domain Detector 79
4.3.4 SE Adapters 81
4.4 Domain-Universal Detection by Domain Attention 82
4.4.1 Simple Domain-Universal Detector 82

4.4.2 Domain-Attentive Universal Detector . . . 83
4.4.3 Universal SE Adapter Bank . . . 84
4.4.4 Domain Attention . . . 85
4.5 Experiments . . . 86
4.5.1 Datasets and Evaluation . . . 87
4.5.2 Single-Domain Detection . . . 87
4.5.3 Multi-Domain Detection . . . 88
4.5.4 Effect of the number of SE adapters . . . 89
4.5.5 Results on the full benchmark . . . 91
4.5.6 Official Evaluation . . . 93
4.6 Conclusion . . . 95
4.7 Acknowledgements . . . 95

Chapter 5 Learning Complexity-Aware Cascades . . . 97
5.1 Introduction . . . 98
5.2 Related Work . . . 101
5.3 Complexity-Aware Cascade . . . 104
5.3.1 AdaBoost . . . 104
5.3.2 Complexity-Aware Learning . . . 105
5.3.3 Embedded Cascade . . . 107
5.3.4 Cascade Boosting . . . 107
5.3.5 Properties . . . 110
5.3.6 Complexity Loss . . . 112
5.3.7 Weak Learners . . . 113
5.3.8 Bootstrapping and Thresholds . . . 116
5.4 Pedestrian Detector Design . . . 117
5.4.1 Heterogeneous Features . . . 117
5.4.2 Feature Complexity . . . 120
5.4.3 Embedding Large CNN Models . . . 122
5.5 Experiments . . . 125
5.5.1 Experimental Setting . . . 125
5.5.2 Homogeneous Cascade Comparison . . . 126
5.5.3 CompACT Cascade Configuration . . . 127
5.5.4 Benefits of optimal accuracy/complexity trade-off . . . 129
5.5.5 Complexity Restricted vs. Sensitive Trees . . . 131
5.5.6 Embedding Large CNN models . . . 132
5.5.7 Pedestrian Detection on Caltech . . . 135
5.5.8 Pedestrian Detection on KITTI . . . 137
5.5.9 Pedestrian Detection on CityPersons . . . 139
5.5.10 CompACT as Attention Mechanism . . . 139
5.6 Conclusion . . . 141
5.7 Acknowledgements . . . 141

Chapter 6 Learning Low-Precision Neural Networks . . . 142
6.1 Introduction . . . 143
6.2 Related Work . . . 145
6.3 Binary Networks . . . 146
6.3.1 Goals . . . 147
6.3.2 Weight Binarization . . . 147
6.3.3 Binary Activation Quantization . . . 148
6.4 Half-wave Gaussian Quantization . . . 149
6.4.1 ReLU . . . 150
6.4.2 Forward Approximation . . . 150
6.4.3 Backward Approximation . . . 153
6.5 Experimental Results . . . 155
6.5.1 Implementation Details . . . 155
6.5.2 Full-precision Activation Comparison . . . 156
6.5.3 Low-bit Activation Quantization Results . . . 157
6.5.4 Backward Approximations Comparison . . . 159
6.5.5 Bit-width Impact . . . 160
6.5.6 Comparison to the state-of-the-art . . . 161
6.5.7 Results on CIFAR-10 . . . 163
6.6 Conclusion . . . 163
6.7 Acknowledgements . . . 164

Chapter 7 Discussion and Conclusion . . . 165

Bibliography . . . 168

LIST OF FIGURES

Figure 1.1: The difference between object classification and object detection. 2

Figure 2.1: In natural images, objects can appear at very different scales, as illustrated by the yellow bounding boxes. A single receptive field, such as that of the RPN [140] (shown in the shaded area), cannot match this variability. 13

Figure 2.2: Different strategies for multi-scale detection. The length of model template represents the template size. 15

Figure 2.3: Proposal sub-network of the MS-CNN. The bold cubes are the output tensors of the network. $h \times w$ is the filter size, c the number of classes, and b the number of bounding box coordinates. 17

Figure 2.4: Object detection sub-network of the MS-CNN. "trunk CNN layers" are shared with proposal sub-network. W and H are the width and height of the input image. The green (blue) cubes represent object (context) region pooling. 21

Figure 2.5: Proposal recall on the KITTI validation set (moderate). "hXXX" refers to input images of height "XXX". "mt" indicates multi-task learning of proposal and detection sub-networks. 25

Figure 2.6: Proposal performance comparison on KITTI validation set (moderate). The first row is proposal recall curves and the second row is recall v.s. IoU for 100 proposals. 26

Figure 2.7: Comparison to the state-of-the-art on KITTI benchmark test set (moderate). 28

Figure 2.8: Comparison to the state-of-the-art on Caltech. 30

Figure 3.1: (a) and (b) detections by object detectors of increasing qualities, and (c) examples of increasing quality. 33

Figure 3.2: Bounding box localization, classification loss and detection performance of object detectors of increasing IoU threshold u. 34

Figure 3.3: The architectures of different frameworks. "I" is input image, "conv" backbone convolutions, "pool" region-wise feature extraction, "H" network head, "B" bounding box, and "C" classification. "B0" is proposals in all architectures. 39

Figure 3.4: IoU histograms of training samples of each cascade stage. The distribution of the 1st stage is the RPN output. Shown in red are the percentage of positives for the corresponding IoU threshold. 44

Figure 3.5: Distribution of the distance vector Δ of (3.4) (without normalization) at different cascade stages. Top: plot of (δ_x, δ_y). Bottom: plot of (δ_w, δ_h). Red dots are outliers for the increasing IoU thresholds of later stages, and the statistics shown are obtained after outlier removal. 47

Figure 3.6: Architectures of the Mask R-CNN (a) and three Cascade Mask R-CNN strategies for instance segmentation (b)-(d). Beyond the definitions of Figure 3.3, "S" denotes a segmentation branch. Note that segmentations branches do not necessarily share heads with the detection branch. 51

Figure 3.7: (a) detection performance of individually trained detectors, with their own proposals (solid curves) or Cascade R-CNN stage proposals (dashed curves). (b) results of adding ground truth to the proposal set. 55
Figure 3.8: Detection performance of all Cascade R-CNN detectors at all cascade stages. 56
Figure 3.9: (a) localization performance of *iterative BBox* and Cascade R-CNN regressors. (b) detection performance of the individual classifiers of the *integral loss* detector. 57

Figure 4.1: Samples of our universal object detection benchmark. 74
Figure 4.2: Multi-domain and universal object detectors for three domains. "D" is the domain, "O" the output, "A" domain-specific adapter, and "DA" the proposed domain attention module. The blue color and the DA are domain-universal, but the other colors domain-specific. 75
Figure 4.3: The activation statistics of all single-domain detectors. 80
Figure 4.4: (a) block diagram of SE adapter and (b) SE adapter bank. 81
Figure 4.5: The block diagram (top) and the detailed view (down) of the proposed domain adaptation module. 83
Figure 4.6: Soft assignments across SE units for all datasets. 91

Figure 5.1: Decision tree of depth 2. Circles represent classifier nodes, squares terminal nodes. $\phi(x, v)$ is the feature of x used at node v, θ a threshold. 113
Figure 5.2: Sample of the feature channels generated on a pedestrian image. 117
Figure 5.3: Eight 2×2 checkerboard-like filters used in this work. Red (Green) is used to represent value +1 (-1). 119
Figure 5.4: CNN architecture used to extract small CNN features. 120
Figure 5.5: Cascade accuracy v.s. complexity for different features. 126
Figure 5.6: Stage configuration learned with different trade-off coefficient η. Only one in five stages is shown. 127
Figure 5.7: Accuracy v.s. complexity for different trade-off coefficients η. 128
Figure 5.8: Stage configuration for "Boosting" and "Manual" cascades. 130
Figure 5.9: Comparison to the state-of-the-art on Caltech (reasonable). 136
Figure 5.10: Processing time spent per Caltech test image. 139

Figure 6.1: Forward and backward functions for binary *sign* (left) and half-wave Gaussian quantization (right) activations. 148
Figure 6.2: Dot-product distributions on different layers of AlexNet with binary weights and quantized activations (100 random images). 152
Figure 6.3: Backward piece-wise activation functions of clipped ReLU and log-tailed ReLU. 154
Figure 6.4: The error curves of training (thin) and test (thick) for $sign(x)$ and $Q(x)$ (HWGQ) activation functions. 158
Figure 6.5: The error curves of training (thin) and test (thick) for alternative backward approximations. 159

LIST OF TABLES

Table 2.1: Parameter configurations of the different models. 24
Table 2.2: Detection recall of the various detection layers on KITTI validation set (car), as a function of object hight in pixels. 25
Table 2.3: Results on the KITTI validation set. "hXXX" indicates an input of height "XXX", "2x" deconvolution, "ctx" context encoding, and "c" dimensionality reduction convolution. In columns "Time" and "# params", entries before the "/" are for car model and after for pedestrian/cyclist model. 27
Table 2.4: Results on the KITTI benchmark test set (only published works shown). . . 29

Table 3.1: Comparison of the Cascade R-CNN with *iterative BBox* and *integral loss* detectors. 57
Table 3.2: Stagewise performance of the Cascade R-CNN. $\overline{1 \sim 3}$ indicates an ensemble result, obtained by averaging the three classifier probabilities for 3rd stage proposals. 58
Table 3.3: Ablation experiments. "IoU↑" indicates increasing IoU thresholds, "*update statistics*" updating regression statistics, and "*stage loss*" weighting of stage losses. 59
Table 3.4: The impact of the number of stages in Cascade R-CNN. 60
Table 3.5: Performance of state-of-the-art *single-model* detectors on COCO `test-dev`. Entries denoted by * and * use enhancements at training and inference, respectively. 61
Table 3.6: Performance of Cascade R-CNN implementations with multiple detectors. All speeds are reported per image on a single Titan Xp GPU. 62
Table 3.7: Performance of various implementations of the Cascade R-CNN with the FPN detector on Detectron, using the `1x` schedule. 64
Table 3.8: Proposal recall of Cascade R-CNN stages. 65
Table 3.9: The instance segmentation comparison among three strategies of the Cascade Mask R-CNN. 66
Table 3.10: Performance of the Cascade Mask R-CNN on multiple backbone networks on COCO 2017 `val`. * and * denotes enhancement techniques at training and inference, respectively, as in [49]. 67
Table 3.11: Detection results on PASCAL VOC 2007 `test`. 68
Table 3.12: MS-CNN detection results for the car class on KITTI test set. 69
Table 3.13: MS-CNN detection results on CityPersons validation set. 69
Table 3.14: MS-CNN Detection results on WiderFace validation set. 70

Table 4.1: The dataset details, the domain-specific hyperparameters and the performance of the single-domain detectors. "T/V/T" means train/val/test, "size" the shortest side of inputs, *BS* RPN batch size, and S/R anchor "scales/aspect ratios". 86

Table 4.2: The comparison on multi-domain detection. † denotes fixed assignment. "time" is the relatively run-times on the five datasets when the domain is unknown. 88
Table 4.3: The effect of SE adapters number. 88
Table 4.4: Overall results on the full universal object detection benchmark (11 datasets). "ada" indicates adapter, "WF" WiderFace, "DL" DeepLesion, "WC" Watercolor. 90
Table 4.5: The comparison on VOC 2007 `test`. ‡/† denotes with COCO `trainval`/`val` 93
Table 4.6: The comparison on WiderFace `Val`. 93
Table 4.7: The comparison on KITTI `test` set of car. 94
Table 4.8: Sensitivity at 4 FPs per image on DeepLesion `test` set. 94
Table 4.9: The comparison on Clipart, Watercolor and Comic `test` set. 95

Table 5.1: Feature complexity and processing unit 121
Table 5.2: Comparison to "Boosting" and "Manual" cascades. 131
Table 5.3: Comparison of complexity restricted and sensitive trees. 131
Table 5.4: Faster-RCNN baseline results. 132
Table 5.5: Embedding of large CNNs in CompACT cascades. "+" denotes additional time, after embedding the CNN, over CompACT. 133
Table 5.6: Pedestrian detection mAP comparison on KITTI. 137
Table 5.7: The results on CityPersons validation set. 138

Table 6.1: Full-precision Activation Comparison for AlexNet. 157
Table 6.2: Low-bit Activation Comparison. 157
Table 6.3: Backward Approximations Comparison. 159
Table 6.4: Bit-width Comparison of Activation Quantization. 160
Table 6.5: HWGQ implementation of various popular networks. 161
Table 6.6: Comparison to state-of-the-art low-precision methods. Top-1 gap to the corresponding full-precision network is also reported. 162
Table 6.7: The results on CIFAR-10. The bit width before and after "+" is for weights and activations respectively. 163

ACKNOWLEDGEMENTS

Finishing the PhD degree would be impossible for me, without the help, support, discussion, and company from many people. I would like to express my sincere gratitude to my family, advisor, thesis committee members, mentors, collaborators, colleagues, friends, etc.

At first, I would like to thank my PhD advisor, Professor Nuno Vasconcelos. It was my great honor to work under his supervision. His critical thinking, research philosophy, technical writing/presentation, etc., have reshaped my research mindset. I would also like to thank Professors Kenneth Kreutz-Delgado, David Kriegman, Truong Nguyen, and Zhuowen Tu for being my committee members and their insightful suggestions and comments.

I was fortunate to work as a research intern in multiple industry research labs. Those research experiences are insightful and unique. I would like to thank Dr. Quanfu Fan and Rogerio Feris of IBM Watson Research, for offering me my first industry research intern experience and their support and help for my project. I would also like to thanks Dr. Xiaodong He and Jian Sun for being my mentors at Microsoft Research. In addition, I am also greatly appreciated to work with Dr. Kaiming He, Ross Girshick and Piotr Dollar at Facebook AI Research. They are the people I look up to in the research area of object detection. Their vision and rigorous experimental validation have deep effect on me as a researcher.

I would like to thank my undergraduate advisor, professor Qi Zhang, for introducing me into this fantastic field of computer vision in the first place, and my career path has been completely changed. I also want to thank my intern advisors, Stan Z. Li, Zhen Lei, Dong Yi, and Shengcai Liao, for providing a rigorous academic training to me at Institute of Automation, Chinese Academy of Science. And my gratitude also goes to my close collaborator, Longyin Wen, for those countless hours of working together.

I also want to thank my colleagues of SVCL who have overlap with me, Dashan Gao, Mohammad Saberian, Jose Costa Pereira, Weixin Li, Mandar Dixit, Can Xu, Song Lu, Si Chen, Bo Liu, Yingwei Li, Pedro Morgado, Yi Li, Pei Wang, John Ho, Gina Wu, Brandon Leung,

Xudong Wang, Gautam Nain, Yang Du, Xiangyun Zhao, Xin Dong, etc. Without their company, the research and life during my PhD study will be much more difficult. Among them, I am particularly thankful to Mohammad Saberian who was very helpful when I started to work at SVCL on object detection, Dashan Gao and Xudong Wang who are co-authors with me, and Jose Costa Pereira, Yingwei Li and Pedro Morgado for maintaining the servers at the lab. I would like to thank also my friends I met at UCSD, Saining Xie, Eshed Ohn-Bar, Yongxi Lu, Yufei Wang, Panqu Wang, Ning Ma, Mengting Wan, Xiaochu Liu, Tong Jiang, Xueying Lyu, Chicheng Zhang, Lidaling Yi, Yundi Zheng, Jiahao Gong and many others, for bringing laughs, happiness and diversity into my life and study.

Outside of UCSD, I am fortunate to meet many friends, who have broaden my horizons, including Xinlei Chen, Haoqi Fan, Chuang Gan, Zhe Gan, Wenbo Li, Sifei Liu, Jiasen Lu, Xi Peng, Joshua San Miguel, Shenlong Wang, Xiaolong Wang, Fan Wei, Yuxin Wu, Cihang Xie, Junjie Yan, Jianwei Yang, Zhengheng Yang, Xi Yin, Licheng Yu, Zhiwei Zhang, Ziyu Zhang, Yuyin Zhou, Bohan Zhuang, and many others.

Finally, and most importantly, I would like to thank my family. I am very grateful that my parents were fully supportive that I pursue my PhD degree. I also owe my grandparents gratitude, who were my guardians when I was a child. I also want to thank my younger brother, who is the company of my family when I am thousands of miles away home. I am deeply thank for their unconditional love and support.

Chapter 2 is, in full, based on the material as it appears in the publication of “A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection”, Zhaowei Cai, Quanfu Fan, Rogerio Feris, and Nuno Vasconcelos, In *Proceedings of 14th European Conference on Computer Vision* (ECCV), 2016. The dissertation author was the primary investigator and author of this material.

Chapter 3 is, in full, based on the material as it appears in the publication of “Cascade R-CNN: High Quality Object Detection and Instance Segmentation”, Zhaowei Cai, and Nuno

Vasconcelos, to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2020. The dissertation author was the primary investigator and author of this material.

Chapter 4 is, in full, based on the material as it appears in the publication of "Towards Universal Object Detection by Domain Attention", Xudong Wang, Zhaowei Cai, Dashan Gao, and Nuno Vasconcelos, In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was a joint primary investigator and author of this material.

Chapter 5 is, in full, based on the material as it appears in the publication of "Learning Complexity-Aware Cascades for Pedestrian Detection", Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2019. The dissertation author was the primary investigator and author of this material.

Chapter 6 is, in full, based on the material as it appears in the publication of "Deep Learning with Low Precision by Half-wave Gaussian Quantization", Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos, In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2017. The dissertation author was the primary investigator and author of this material.

VITA

2011	B. S. in Automation, Dalian Maritime University, China
2019	M. S. in Electrical Engineering (Signal and Image Processing), University of California San Diego, United States
2019	Ph. D. in Electrical Engineering (Signal and Image Processing), University of California San Diego, United States

PUBLICATIONS

Zhaowei Cai, and Nuno Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation", to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2020.

Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, "Learning Complexity-Aware Cascades for Pedestrian Detection", to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2019.

Xudong Wang, **Zhaowei Cai**, Dashan Gao, and Nuno Vasconcelos, "Towards Universal Object Detection by Domain Attention", In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019.

Zhaowei Cai, and Nuno Vasconcelos, "Cascade R-CNN: Delving into High Quality Object Detection", In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2018.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos, "Deep Learning with Low Precision by Half-wave Gaussian Quantization", In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2017.

Zhaowei Cai, Quanfu Fan, Rogerio Feris, and Nuno Vasconcelos, "A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection", In *Proceedings of 14th European Conference on Computer Vision* (ECCV), 2016.

Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, "Learning Complexity-Aware Cascades for Deep Pedestrian Detection", In *Proceedings of IEEE International Conference on Computer Vision* (ICCV), 2015.

ABSTRACT OF THE DISSERTATION

Towards Universal Object Detection

by

Zhaowei Cai

Doctor of Philosophy in Electrical Engineering (Signal and Image Processing)

University of California San Diego, 2019

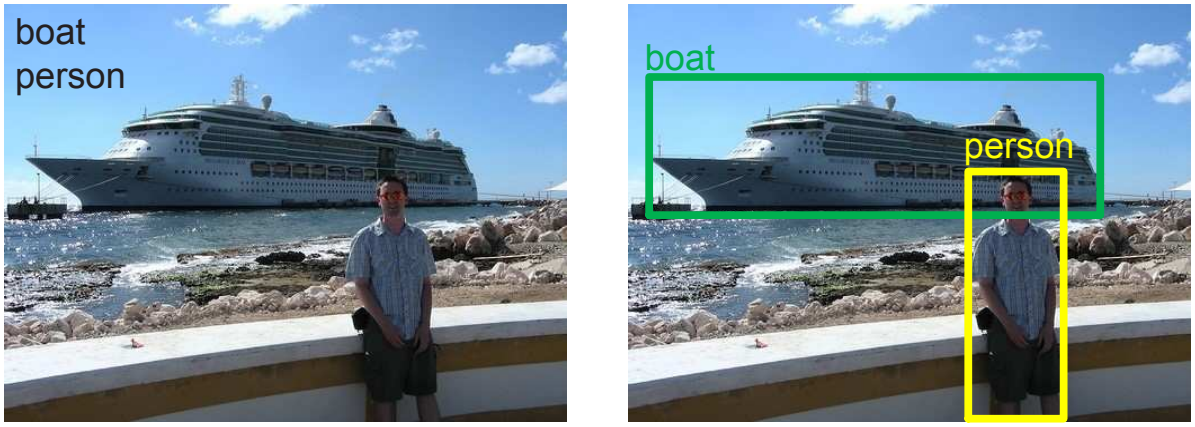
Professor Nuno Vasconcelos, Chair

Object detection is one of the most important and challenging research topics in computer vision. It is playing an important role in our everyday life and has many applications, e.g. surveillance, autonomous driving, robotics, drone, medical imaging, etc. The ultimate goal of object detection is a universal object detector that can work very well in any case under any condition like human vision system. However, there are multiple challenges on the universality of object detection, e.g. scale-variance, high-quality requirement, domain shift, computational constraint, etc. These will prevent the object detector from being widely used for various scales of objects, critical applications requiring extremely accurate localization, scenarios with changing domain priors, and diverse hardware settings. To address these challenges, multiple solutions

have been proposed in this thesis. These include an efficient multi-scale architecture to achieve scale-invariant detection, a robust multi-stage framework effective for high-quality requirement, a cross-domain solution to extend the universality over various domains, and a design of complexity-aware cascades and a novel low-precision network to enhance the universality under different computational constraints. All these efforts have substantially improved the universality of object detection, and the advanced object detector can be applied to broader environments.

Chapter 1

Introduction



(a) Object Classification (b) Object Detection

Figure 1.1: The difference between object classification and object detection.

Computer vision is a scientific field that studies the acquisition, extraction, processing, analysis, interpretation and understanding of digital images in order to produce numerical or symbolic information, e.g. in the forms of decisions. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory [111, 39]. It is an interdisciplinary scientific filed with cognitive science, robotics, machine learning, statistics, etc. Its goal is to simulate the human vision system by automatic machines. It has multiple sub-domains, including visual recognition, action understanding, image retrieval, video tracking, segmentation, 3D reconstruction, etc.

Computer vision tasks have achieved great successes in recent years, powered by the recent great successes of deep learning [86, 150, 152, 61]. These are enabled by big data, e.g. large scale annotated dataset, and big computation, e.g. distributed or parallel computations. The feature representation learning by deep neural network is much more representative and powerful than the hand-crafted feature extraction, e.g. HOG [26] and SIFT [109]. The substantial progresses in computer vision make it possible to deploy computer vision algorithms into critical applications, e.g. autonomous driving, medical imaging diagnosis, surgery and service robotics, etc.

1.1 Object Detection

Object detection is one of the most important and challenging research topics in computer vision. What is object detection? Object classification is probably a more common problem. Given an image, the task is to answer the question of "what". For example, in Figure 1.1 (a), there are a person and a boat. In addition to the question of "what", object detection also needs to answer the "where" question. For example, in Figure 1.1 (b), knowing there are a person and a boat is not enough. We also want to where they are.

Object detection is important. On one hand, it has many practical applications, e.g. surveillance, autonomous robot, drone, medical imaging, etc. For example, in autonomous driving, object detection is used to localize objects, understand surrounding environments, and help to make safe decisions. It can also help to detect the lesion in medical CT scans to alleviate the burden of radiologists. On the other hand, object detection is a upstream task in computer vision, supporting many downstream tasks, like visual question answering, captioning, visual navigation, robot grasping, knowledge graph, pose estimation, etc. For example, object detection can not only help the robot arm to accurately grasp the objects in the physical world, but also enable bots to better understand the image semantics and provide more precise answers to the questions asked by the users. Hence, the advancement of object detection is necessary. It will benefit many other sub-domains of computer vision and improve practical computer vision systems to better serve our everyday life.

1.2 Challenges on Object Detection Universality

Despite object detection has made huge progresses in recent years [51, 59, 50, 140, 99, 58], there are still many unresolved challenges. For example, the object detector is usually required to produce accurate detection results of various categories, scales, aspect ratios, etc, and sometimes even with serious lighting changes, occlusion, background distraction, etc. It is very challenging

to develop a universal object detector that can work well in any case under any condition like human visual system. In specific, the universality challenge has multiple aspects, including scale-variance, high-quality requirement, domain shift, device and computation constraints, etc.

First, scale invariance is an important requisite for effective object detection, since the objects could be presented with huge scale variation, e.g. from 5×5 pixels to 1000×1000 pixels, in a typical image. A detector should detect all objects without scale bias. This is important for practical application of object detection. For example, in the vision system of autonomous vehicle, where the further objects are usually visually smaller, if smaller (further) objects can be successfully detected, the control system will have more time to respond. However, the detection of small objects is much more difficult than that of large objects, reducing the universality of object detection in scale. There are mainly two reasons responsible for this: 1) the feature extraction is not scale invariant; and 2) it is nontrivial to design an effective and efficient multi-scale detection framework. To address each of them, before the resurgence of deep learning, the solutions are to manually design better feature extraction, e.g. HOG [26] and SIFT [109], and to use the image pyramid strategy, respectively. However, the handcrafted feature extraction is not very representative and the image pyramid is very time-consuming. Although deep neural networks can offer much more powerful feature representations, the design of efficient multi-scale detection system within neural network is not trivial.

Second, detection quality is an important indicator of how good an object detector is. Higher detection quality means the detector can produce detection results that have tighter coverage with the ground truths. The tightness is evaluated by a metric, called intersection over union (IoU), between a detection and the corresponding ground truth. If the IoU is above a threshold, e.g. 0.5, the detection is a success, otherwise, a failure. However, the importance of high-quality detection is generally neglected in the area of object detection. This is partially because the detection evaluation metrics have historically placed greater emphasis on the low-quality detection regime. Many object detection datasets, including the very popular PASCAL

VOC [36], ImageNet [143], Caltech Pedestrian[32], etc., use IoU threshold of 0.5 in evaluation, which is a very loose requirement. Nevertheless, it is important to have high quality detections, e.g. using IoU threshold of 0.7. For example, if the detection of a pedestrian is not highly accurate, the vision system of an autonomous vehicle could mispredict the pedestrian crossing the street as walking on the sidewalk. Another example is from robotics, e.g. surgery or service robots. Any visual detection in surgery robot is required to be highly precise. Nobody wants his/her organs or tissues to be mislocalized or miscut in any way. The consequences of low quality detection in these critical applications are very serious. The current object detectors are usually good at low-quality detection, but not high-quality detection. It is very challenging to design quality-universal object detection system.

Third, existing detectors are usually *domain-specific*, e.g. trained and tested on a single dataset, instead of *domain-universal*. This is partially due to the fact that object detection datasets are diverse and there are nontrivial domain shifts among them. In general, high detection performance requires a detector specialized on the target dataset. This poses a significant problem for practical applications, which are not usually restricted to any one of the domains. For example, an autonomous robot can go into any environments, with changing illumination, background, weather, etc. The *domain-specific* detector probably will fail in this case. Hence, there is a need for systems capable of detecting objects regardless of the domains. However, developing this domain universal detection system is very challenging, e.g. for more than thousands of different domains, considering that a single model may not have enough capacity to cover the substantial domain shifts and the model size and computation need to be prevented from being increasing too much.

Fourth, object detection is an indispensable module in many computer vision systems, which are frequently required to run on different kinds of hardware. Although the overwhelming popularity of neural networks make the GPU as the most popular hardware, its disadvantages including expensive price, high power usage, large volume, heavy weight, etc., encourage to

use other hardware to meet different kinds of needs, e.g. CPU, FPGA, cellphone, raspberry pi, etc. However, these non-GPU hardware generally has strong constraints on power, computation, memory, etc., which could be orders of lower than those of GPUs. In these cases, the detector usually has to compromise on speed and performance due to the constraints. Therefore, the current object detectors are not universal for different devices or under different constraints. How to improve speed, reduce power consumption and remain accuracy on non-GPU devices are important and challenging.

1.3 Contributions of the Thesis

In this thesis, we mainly try to tackle the four challenges mentioned above, scale, quality, domain and complexity constraint, and to advance the universality of object detection.

1.3.1 An Efficient Framework for Multi-Scale Detection

Although deep neural networks have achieved great successes in object detection [51, 59, 50, 140, 99, 58], how to design effective and efficient multi-scale detection system in neural networks is still under exploration. Our goal is to develop a new architecture of convolutional neural network for multi-scale object detection, improving the detection universality across scales. The scale-invariant detection is traditionally addressed by operating a scale-specific detector on a pyramid of images: the larger (smaller) objects are detected on the coarser (finer) resolution image. The outputs on all pyramid images are grouped together as the final detection results. However, this strategy is not desirable, especially for neural networks, because it requires to run the expensive network multiple times for all images in the pyramid. Instead, we avoid the problem by resorting to a pyramid of features, as an approximation to image pyramid. The efficient feature pyramid will be achieved by forwarding the original input image to the expensive network only once, and then multiple scale-specific light-weight detectors run on corresponding

feature scale. These complementary detectors are grouped together as a final strong multi-scale object detector. This feature pyramid strategy can successfully avoid the repetitive expensive network computations for image pyramid, and thus are much more efficient. The experiments have shown that it can significantly improve the detection performance on small objects, enabling better scale-invariant detection. Thus, the universality over scale is advanced.

1.3.2 An Effective Architecture for High-Quality Detection

In object detection, there are many "close" false positives presented in an image, corresponding to "close but not correct" bounding boxes. An effective detector must find all true positives, while suppressing these close false positives. This requirement makes detection more difficult than object recognition. Unlike the latter, an intersection over union (IoU) threshold is required to define positives/negatives in object detection, which as well defines its *quality*. While the threshold is typically set at the value of 0.5, this is a very loose requirement for positives. The resulting detectors frequently produce noisy (low quality) detections. However, detection performance usually degrades when using higher training IoU thresholds. This is denoted the *paradox* of high quality detection and has two main causes: 1) overfitting, due to exponentially vanishing positive training sets for large thresholds, and 2) inference-time mismatch between the IoUs for which the detector is optimal and those of the test hypotheses. A multi-stage object detection architecture, the Cascade R-CNN, is proposed to address these problems. It consists of a sequence of detectors trained with increasing IoU thresholds, so that deeper stages are more selective against close false positives. The detectors are trained sequentially, using the output of a detector as the training set for the subsequent one. This resampling operation progressively improves hypotheses quality, guaranteing that all detectors have a positive training set of equivalent size and thereby eliminating overfitting problem. The use of the same cascade procedure at inference time then eliminates the quality mismatch between the hypotheses and the detector of each stage. The Cascade R-CNN has been shown to significantly improve the high-quality object

detection, and thus advance the detection universality over quality.

1.3.3 A Novel Design for Domain-Universal Detection

Despite increasing efforts on multi-domain/universal representations for visual recognition, few have addressed object detection. In this thesis, we developed a few systems for multi-domain and domain-universal object detection, capable of working on various image domains, e.g. from human faces and traffic signs to medical CT images. In the multi-domain system, a new family of domain adapter for object detection was introduced which is effective and efficient, based on the principles of squeeze and excitation. However, this multi-domain detection system still requires domain prior, knowing which domain to work on in advance. Unlike the multi-domain system, the domain-universal system does not require prior knowledge of the domain of interest. This is achieved by the introduction of a new domain-attention mechanism. In this domain-universal system, all parameters and computations are shared across domains, and a single network processes all domains all the time. The experiments have shown that these solutions can substantially extend the detector universality across a large number of domains with small amount of increases in overall model size and computation.

1.3.4 Optimal Detector Solutions under Complexity Constraint

How to improve speeds and reduce power consumption are necessary for object detection. We tried to tackle these challenges from two aspects: the design of a novel complexity-aware detector cascade and learning efficient low-precision networks. They improve the detection universality across computation constraints and hardware.

Cascade object detector is one of the most popular detection frameworks in the history, especially before the resurgence of deep learning. We investigated the design of complexity-aware cascaded detectors, combining features of very different complexities. This cascade procedure is

designed, by formulating cascade learning as the Lagrangian optimization of a risk that accounts for both accuracy and complexity. A boosting algorithm, denoted as *complexity aware cascade training* (CompACT), is then derived to solve this optimization. CompACT cascades are shown to seek an optimal trade-off between accuracy and complexity by pushing features of higher complexity to the later cascade stages, where only a few difficult candidate patches remain to be classified. This enables the use of features of vastly different complexities in a single detector. In result, the feature pool can be expanded to features previously impractical for cascade design, such as the responses of a deep convolutional neural network (CNN). This leads to mixed CPU/GPU solutions, that provide practitioners with flexible families of models, ranging from cheap and energy efficient CPU-based cascades, to cascades using GPU stages that emphasize recognition accuracy. This enables accurate cascade at fairly fast speeds.

Current cutting-edge computer vision systems extensively rely on deep neural networks, which are heavy in terms of computation and memory, and require specialized expensive hardware, e.g. GPUs. Substantial speed-ups will be made possible if, both activations and weights of a network are binarized or quantized to low-bit, leveraging the property that the expensive float-point computation of dot-product can be replaced by efficient bit operations. However, this is challenging and suffers from substantial accuracy loss, especially when the activations are quantized. Thus, we investigated the problem of quantizing the activations for the speeds-up of neural networks. In previous binary quantization approaches, this problem consists of approximating a classical non-linearity, the hyperbolic tangent, by two functions: a piecewise constant *sign* function, which is used in feedforward network computations, and a piecewise linear *hard tanh* function, used in the backpropagation step during network learning. Instead, we considered to approximate the widely used ReLU non-linearity. An half-wave Gaussian quantizer (HWGQ) is proposed for forward approximation and shown to have efficient implementation, by exploiting the statistics of network activations and batch normalization operations. To overcome the problem of gradient mismatch, due to the use of different forward and backward approximations, several

piece-wise backward approximators are then investigated. The implementation of the resulting quantized network, denoted as HWGQ-Net, is shown to achieve much closer performance to full precision networks. This network quantization technique can be used to significantly improve the running speeds and save power for object detectors based on deep neural networks, enabling them to run on non-GPU devices, e.g. CPU or FPGA.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 describes an efficient and effective architecture of convolutional neural network for multi-scale object detection, which is shown to enable better scale-invariant detection. In Chapter 3, we review the novel multi-stage architecture for high-quality object detection, denoted as Cascade R-CNN, and show that this simple design can achieve very robust and strong high-quality detection. Chapter 4 introduces a few solutions for multi-domain and domain-universal object detection, to improve the detection universality across different domains. Chapter 5 investigates the design of complexity-aware cascaded detectors, combining features of very different complexities. The proposed cascades are shown to seek an optimal trade-off between accuracy and complexity by pushing features of higher complexity to the later cascade stages, where only a few difficult candidate patches remain to be classified. In Chapter 6, we present a new quantization technique for low-precision networks, which can significantly save computation and memory with minor accuracy drop, enabling running expensive neural networks on non-GPU devices, e.g. CPU or FPGA. Finally, the thesis is summarized and concluded in Chapter 7.

Chapter 2

Multi-Scale Object Detection

2.1 Introduction

Classical object detectors, based on the sliding window paradigm, search for objects at multiple scales and aspect ratios. While real-time detectors are available for certain classes of objects, e.g. faces or pedestrians [161, 29], it has proven difficult to build detectors of multiple object classes under this paradigm. Recently, there has been interest in detectors derived from deep convolutional neural networks (CNNs) [51, 50, 16, 59, 46]. While these have shown much greater ability to address the multiclass problem, less progress has been made towards the detection of objects at multiple scales. The R-CNN [51] samples object proposals at multiple scales, using a preliminary attention stage [158], and then warps these proposals to the size (e.g. 224×224) supported by the CNN. This is, however, very inefficient from a computational standpoint. The development of an effective and computationally efficient region proposal mechanism is still an open problem. The more recent Faster-RCNN [140] addresses the issue with a region proposal network (RPN), which enables end-to-end training. However, the RPN generates proposals of multiple scales by sliding a fixed set of filters over a fixed set of convolutional feature maps. This creates an inconsistency between the sizes of objects, which are variable, and filter receptive fields, which are fixed. As shown in Figure 2.1, a fixed receptive field cannot cover the multiple scales at which objects appear in natural scenes. This compromises detection performance, which tends to be particularly poor for small objects, like that in the center of Figure 2.1. In fact, [50, 16, 140] handle such objects by upsampling the input image both at training and testing time. This increases the memory and computation costs of the detector.

This work proposes a unified multi-scale deep CNN, denoted the *multi-scale CNN* (MS-CNN), for fast object detection. Similar to [140], this network consists of two sub-networks: an object proposal network and an accurate detection network. Both of them are learned end-to-end and share computations. However, to ease the inconsistency between the sizes of objects and receptive fields, object detection is performed with multiple output layers, each focusing

Figure 2.1: In natural images, objects can appear at very different scales, as illustrated by the yellow bounding boxes. A single receptive field, such as that of the RPN [140] (shown in the shaded area), cannot match this variability.

on objects within certain scale ranges (see Figure 2.3). The intuition is that lower network layers, such as "conv-3," have smaller receptive fields, better matched to detect small objects. Conversely, higher layers, such as "conv-5," are best suited for the detection of large objects. The complimentary detectors at different output layers are combined to form a strong multi-scale detector. This is shown to produce accurate object proposals on detection benchmarks with large variation of scale, such as KITTI [43], achieving a recall of over 95% for only 100 proposals.

A second contribution of this work is the use of feature upsampling as an alternative to input upsampling. This is achieved by introducing a deconvolutional layer that increases the resolution of feature maps (see Figure 2.4), enabling small objects to produce larger regions of strong response. This is shown to reduce memory and computation costs. While deconvolution has been explored for segmentation [107] and edge detection [173], it is, as far as we know, for the first time used to speed up and improve detection. When combined with efficient context encoding and hard negative mining, it results in a detector that advances the state-of-the-art detection on the KITTI [43] and Caltech [32] benchmarks. Without image upsampling, the MS-CNN achieves speeds of 10 fps on KITTI (1250×375) and 15 fps on Caltech (640×480) images.

2.2 Related Work

One of the earliest methods to achieve real-time detection with high accuracy was the cascaded detector of [161]. This architecture has been widely used to implement sliding window detectors for faces [161, 8], pedestrians [29, 11] and cars [122]. Two main streams of research have been pursued to improve its speed: fast feature extraction [161, 29] and cascade learning [8, 145, 11]. In [161], a set of efficient Haar features was proposed with recourse to integral images. The aggregate feature channels (ACF) of [29] made it possible to compute HOG features at about 100 fps. On the learning front, [8] proposed the soft-cascade, a method to transform a classifier learned with boosting into a cascade with certain guarantees in terms of false positive and detection rate. [145] introduced a Lagrangian formulation to learn cascades that achieve the optimal trade-off between accuracy and computational complexity. [11] extended this formulation for cascades of highly heterogeneous features, ranging from ACF set to deep CNNs, with widely different complexity. The main current limitation of detector cascades is the difficulty of implementing multiclass detectors under this architecture.

In an attempt to leverage the success of deep neural networks for object classification, [51] proposed the R-CNN detector. This combines an object proposal mechanism [158] and a CNN classifier [86]. While the R-CNN surpassed previous detectors [38, 164] by a large margin, its speed is limited by the need for object proposal generation and repeated CNN evaluation. [59] has shown that this could be ameliorated with recourse to spatial pyramid pooling (SPP), which allows the computation of CNN features once per image, increasing the detection speed by an order of magnitude. Building on SPP, the Fast-RCNN [50] introduced the ideas of back-propagation through the ROI pooling layer and multi-task learning of a classifier and a bounding box regressor. However, it still depends on bottom-up proposal generation. More recently, the Faster-RCNN [140] has addressed the generation of object proposals and classifier within a single neural network, leading to a significant speedup for proposal detection. Another interesting work

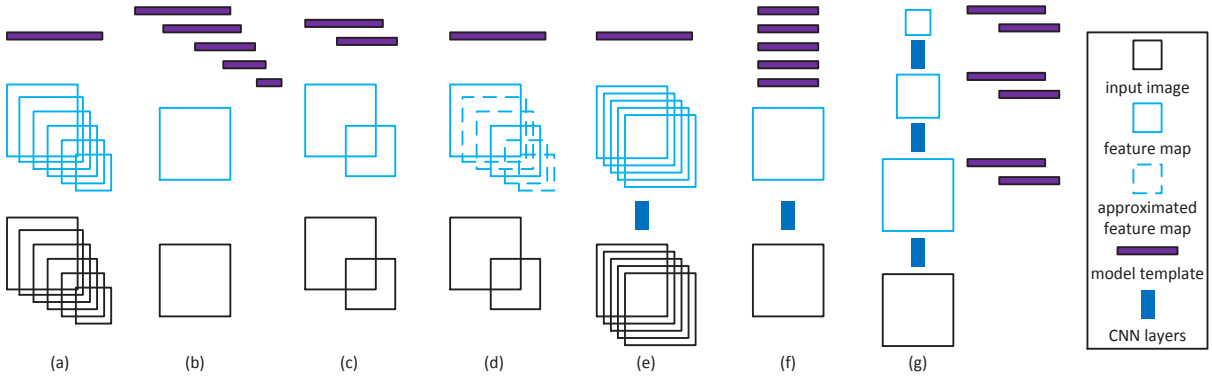


Figure 2.2: Different strategies for multi-scale detection. The length of model template represents the template size.

is YOLO [137], which outputs object detections within a 7×7 grid. This network runs at ∼40 fps, but with some compromise of detection accuracy.

For object recognition, it has been shown beneficial to combine multiple losses, defined on intermediate layers of a single network [152, 88, 107, 173]. GoogLeNet [152] proposed the use of three weighted classification losses, applied at layers of intermediate heights, showing that this type of regularization is useful for very deep models. The deeply supervised network architecture of [88] extended this idea to a larger number of layers. The fact that higher layers convey more semantic information motivated [107] to combine features from intermediate layers, leading to more accurate semantic segmentation. A similar idea was shown useful for edge detection in [173]. Similar to [152, 88, 107, 173], the proposed MS-CNN is learned with losses that account for intermediate layer outputs. However, the aim is not to simply regularize the learning, as in [152, 88], or provide detailed information for higher outputs, as in [107, 173]. Instead, the goal is to produce a strong individual object detector at each intermediate output layer.

2.3 Multi-Scale Object Proposal Network

In this section, we introduce the proposed network for the generation of object proposals.

2.3.1 Multi-Scale Detection Strategies

The coverage of many object scales is a critical problem for object detection. Since a detector is basically a dot-product between a learned template and an image region, the template has to be matched to the spatial support of the object to recognize. There are two main strategies to achieve this goal. The first is to learn a single classifier and rescale the image multiple times, so that the classifier can match all possible object sizes. As illustrated in Figure 2.2 (a), this strategy requires feature computation at multiple image scales. While it usually produces the most accurate detection, it tends to be very costly. An alternative approach is to apply multiple classifiers to a single input image. This strategy, illustrated in Figure 2.2 (b), avoids the repeated computation of feature maps and tends to be efficient. However, it requires an individual classifier for each object scale and usually fails to produce good detectors. Several approaches have been proposed to achieve a good trade-off between accuracy and complexity. For example, the strategy of Figure 2.2 (c) is to rescale the input a few times and learn a small number of model templates [3]. Another possibility is the feature approximation of [29]. As shown in Figure 2.2 (d), this consists of rescaling the input a small number of times and interpolating the missing feature maps. This has been shown to achieve considerable speed-ups for a very modest loss of classification accuracy [29].

The implementation of multi-scale strategies on CNN-based detectors is slightly different from those discussed above, due to the complexity of CNN features. As shown in Figure 2.2 (e), the R-CNN of [51] simply warps object proposal patches to the natural scale of the CNN. This is somewhat similar to Figure 2.2 (a), but features are computed for patches rather than the entire image. The multi-scale mechanism of the RPN [140], shown in Figure 2.2 (f), is similar to that of Figure 2.2 (b). However, multiple sets of templates of the same size are applied to all feature maps. This can lead to a severe scale inconsistency for template matching. As shown in Figure 2.1, the single scale of the feature maps, dictated by the (228×228) receptive field of the CNN, can be severely mismatched to small (e.g. 32×32) or large (e.g. 640×640) objects. This

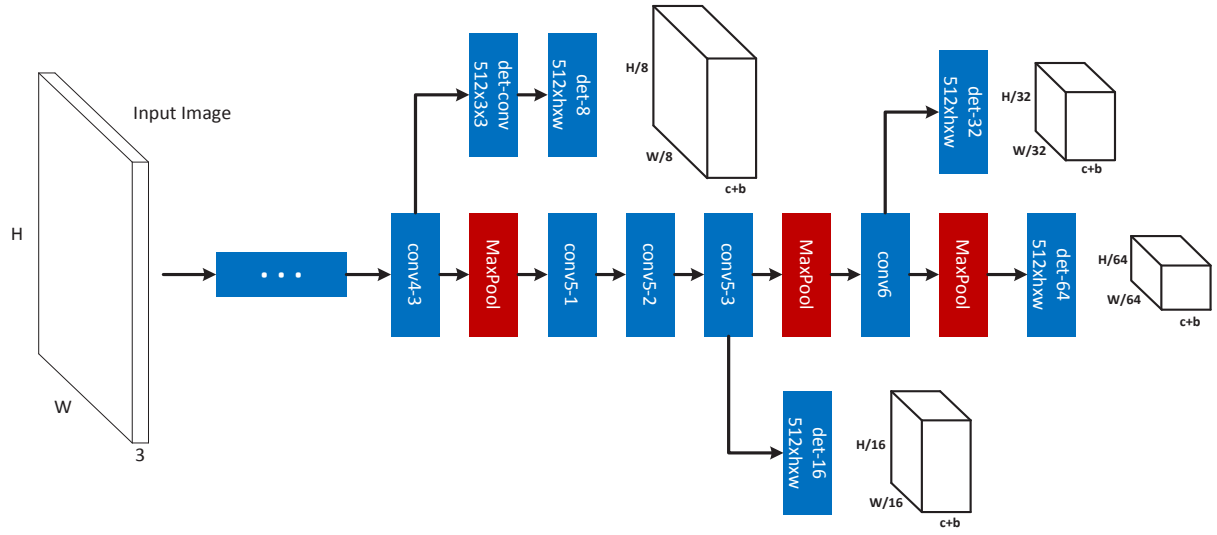


Figure 2.3: Proposal sub-network of the MS-CNN. The bold cubes are the output tensors of the network. $h \times w$ is the filter size, c the number of classes, and b the number of bounding box coordinates.

compromises object detection performance.

Inspired by previous evidence on the benefits of the strategy of Figure 2.2 (c) over that of Figure 2.2 (b), we propose a new multi-scale strategy, shown in Figure 2.2 (g). This can be seen as the deep CNN extension of Figure 2.2 (c), but only uses a single scale of input. It differs from both Figure 2.2 (e) and (f) in that it exploits feature maps of several resolutions to detect objects at different scales. This is accomplished by the application of a set of templates at intermediate network layers. This results in a set of variable receptive field sizes, which can cover a large range of object sizes.

2.3.2 Architecture

The detailed architecture of the MS-CNN proposal network is shown in Figure 2.3. The network detects objects through several detection branches. The results by all detection branches are simply declared as the final proposal detections. The network has a standard CNN trunk, depicted in the center of the figure, and a set of output branches, which emanate from different

layers of the trunk. These branches consist of a single detection layer. Note that a buffer convolutional layer is introduced on the branch that emanates after layer "conv4-3". Since this branch is close to the lower layers of the trunk network, it affects their gradients more than the other detection branches. This can lead to some instability during learning. The buffer convolution prevents the gradients of the detection branch from being back-propagated directly to the trunk layers.

During training, the parameters $\mathbf{W}$ of the multi-scale proposal network are learned from a set of training samples $S = \{(X_i, Y_i)\}_{i=1}^{N}$, where X_i is a training image patch, and $Y_i = (y_i, b_i)$ the combination of its class label $y_i \in \{0, 1, 2, \cdots, K\}$ and bounding box coordinates $b_i = (b_i^x, b_i^y, b_i^w, b_i^h)$. This is achieved with a multi-task loss

$$\mathcal{L}(\mathbf{W}) = \sum_{m=1}^{M} \sum_{i \in S^m} \alpha_m l^m(X_i, Y_i | \mathbf{W}), \tag{2.1}$$

where M is the number of detection branches, α_m the weight of loss l^m, and $S = \{S^1, S^2, \cdots, S^M\}$, where S^m contains the examples of scale m. Note that only a subset S^m of the training samples, selected by scale, contributes to the loss of detection layer m. Inspired by the success of joint learning of classification and bounding box regression [50, 140], the loss of each detection layer combines these two objectives

$$l(X, Y | \mathbf{W}) = L_{cls}(p(X), y) + \lambda [y \geq 1] L_{loc}(b, \hat{b}), \tag{2.2}$$

where $p(X) = (p_0(X), \cdots, p_K(X))$ is the probability distribution over classes, λ a trade-off coefficient, $L_{cls}(p(X), y) = -\log p_y(X)$ the cross-entropy loss, $\hat{b} = (\hat{b}_x, \hat{b}_y, \hat{b}_w, \hat{b}_h)$ the regressed bounding box, and

$$L_{loc}(b, \hat{b}) = \frac{1}{4} \sum_{j \in \{x,y,w,h\}} smooth_{L_1}(b_j, \hat{b}_j), \tag{2.3}$$

the smoothed bounding box regression loss of [50]. The bounding box loss is only used for

positive samples and the optimal parameters $\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W})$ are learned by stochastic gradient descent.

2.3.3 Sampling

This section describes the assembly of training samples $S^m = \{S^m_+, S^m_-\}$ for each detection layer m. In what follows, the superscript m is dropped for notional simplicity. An anchor is centered at the sliding window on layer m associated with width and height corresponding to filter size. More details can be found in Table 2.1. A sample X of anchor bounding box b is labeled as positive if $o^* \geq 0.5$, where

$$o^* = \max_{i \in S_{gt}} IoU(b, b_i). \tag{2.4}$$

S_{gt} is the ground truth and IoU the intersection over union between two bounding boxes. In this case, $Y = (y_{i^*}, b_{i^*})$, where $i^* = \arg\max_{i \in S_{gt}} IoU(b, b_i)$ and (X, Y) are added to the positive set S_+. All the positive samples in $S_+ = \{(X_i, Y_i) | y_i \geq 1\}$ contribute to the loss. Samples such that $o^* < 0.2$ are assigned to a preliminary negative training pool, and the remaining samples discarded. For a natural image, the distribution of objects and non-objects is heavily asymmetric. Sampling is used to compensate for this imbalance. To collect a final set of negative samples $S_- = \{(X_i, Y_i) | y_i = 0\}$, such that $|S_-| = \gamma |S_+|$, we considered three sampling strategies: random, bootstrapping, and mixture.

Random sampling consists of randomly selecting negative samples according to a uniform distribution. Since the distribution of hard and easy negatives is heavily asymmetric too, most randomly collected samples are easy negatives. It is well known that hard negatives mining helps boost performance, since hard negatives have the largest influence on the detection accuracy. Bootstrapping accounts for this, by ranking the negative samples according to their objectness scores, and then collecting top $|S_-|$ negatives. Mixture sampling combines the two, randomly sampling half of S_- and sampling the other half by bootstrapping. In our experiments, mixture

sampling has very similar performance to bootstrapping.

To guarantee that each detection layer only detects objects in a certain range of scales, the training set for the layer consists of the subset of S that covers the corresponding scale range. For example, the samples of smallest scale are used to train the detector of "det-8" in Figure 2.3. It is possible that no positive training samples are available for a detection layer, resulting in $|S_-|/|S_+| \gg \gamma$. This can make learning unstable. To address this problem, the cross-entropy terms of positives and negatives are weighted as follows

$$L_{cls} = \frac{1}{1+\gamma}\frac{1}{|S_+|}\sum_{i\in S_+} -\log p_{y_i}(X_i) + \frac{\gamma}{1+\gamma}\frac{1}{|S_-|}\sum_{i\in S_-} -\log p_0(X_i). \tag{2.5}$$

2.3.4 Implementation Details

Data Augmentation: In [50, 59], it is argued that multi-scale training is not needed, since deep neural networks are adept at learning scale invariance. This, however, is not true for datasets such as Caltech [32] and KITTI [43], where object scales can span multiple octaves. In KITTI, many objects are quite small. Without rescaling, the cardinalities of the sets $S_+ = \{S_+^1, S_+^2, \cdots, S_+^M\}$ are wildly varying. In general, the set of training examples of largest object size is very small. To ease this imbalance, the original images are randomly resized to multiple scales.

Fine-tuning: Training the Fast-RCNN [50] and RPN [140] networks requires large amounts of memory and a small mini-batch, due to the large size of the input (i.e. 1000×600). This leads to a very heavy training procedure. In fact, many background regions that are useless for training take substantially amounts of memory. Thus, we randomly crop a small patch (e.g. 448×448) around objects from the whole image. This drastically reduces the memory requirements, enabling four images to fit into the typical GPU memory of 12G.

Learning is initialized with the popular VGG-Net [150]. Since bootstrapping and the multi-task loss can make training unstable in the early iterations, a two-stage procedure is adopted. The first stage uses random sampling and a small trade-off coefficient λ (e.g. 0.05). 10,000

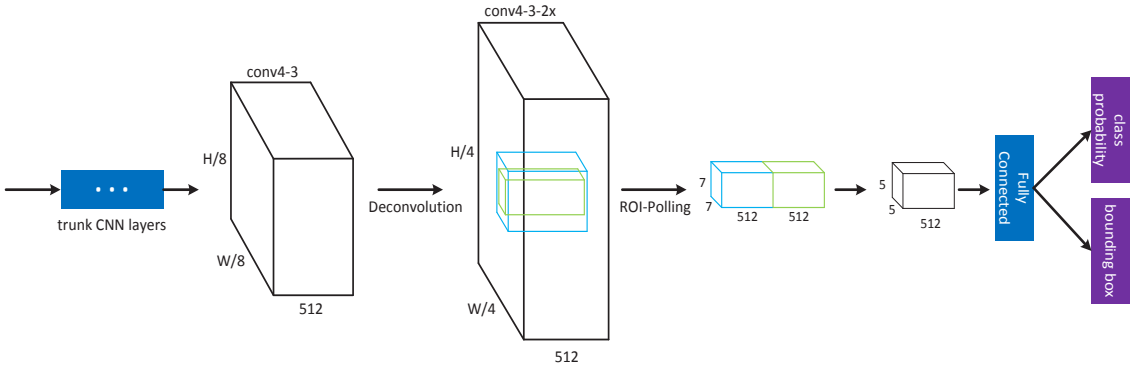


Figure 2.4: Object detection sub-network of the MS-CNN. "trunk CNN layers" are shared with proposal sub-network. W and H are the width and height of the input image. The green (blue) cubes represent object (context) region pooling.

iterations are run with a learning rate of 0.00005. The resulting model is used to initialize the second stage, where random sampling is switched to bootstrapping and $\lambda = 1$. We set $\alpha_i = 0.9$ for "det-8" and $\alpha_i = 1$ for the other layers. Another 25,000 iterations are run with an initial learning rate of 0.00005, which decays 10 times after every 10,000 iterations. This two-stage learning procedure enables stable multi-task training.

2.4 Object Detection Network

Although the proposal network could work as a detector itself, it is not strong, since its sliding windows do not cover objects well. To increase detection accuracy, a detection network is added. Following [50], a ROI pooling layer is first used to extract features of a fixed dimension (e.g. $7\times7\times512$). The features are then fed to a fully connected layer and output layers, as shown in Figure 2.4. A deconvolution layer, described in Section 2.4.1, is added to double the resolution of the feature maps. The multi-task loss of (2.1) is extended to

$$\mathcal{L}(\mathbf{W}, \mathbf{W}_d) = \sum_{m=1}^{M} \sum_{i \in S^m} \alpha_m l^m(X_i, Y_i | \mathbf{W}) + \sum_{i \in S^{M+1}} \alpha_{M+1} l^{M+1}(X_i, Y_i | \mathbf{W}, \mathbf{W}_d), \tag{2.6}$$

where l^{M+1} and S^{M+1} are the loss and training samples for the detection sub-network. S^{M+1} is collected as in [50]. As in (2.2), l^{M+1} combines a cross-entropy loss for classification and a smoothed L_1 loss for bounding box regression. The detection sub-network shares some of the proposal sub-network parameters $\mathbf{W}$ and adds some parameters $\mathbf{W}_d$. The parameters are optimized jointly, i.e. $(\mathbf{W}^*, \mathbf{W}_d^*) = \arg\min \mathcal{L}(\mathbf{W}, \mathbf{W}_d)$. In the proposed implementation, ROI pooling is applied to the top of the "conv4-3" layer, instead of the "conv5-3" layer of [50], since "conv4-3" feature maps performed better in our experiments. One possible explanation is that "conv4-3" corresponds to higher resolution and is better suited for location-aware bounding box regression.

2.4.1 CNN Feature Map Approximation

Input size has a critical role in CNN-based object detection accuracy. Simply forwarding object patches, at the original scale, through the CNN impairs performance (especially for small ones), since the pre-trained CNN models have a natural scale (e.g. 224×224). While the R-CNN naturally solves this problem through warping [51], it is not explicitly addressed by the Fast-RCNN [50] or Faster-RCNN [140]. To bridge the scale gap, these methods simply upsample input images (by ∼2 times). For datasets, such as KITTI [43], containing large amounts of small objects, this has limited effectiveness. Input upsampling also has three side effects: large memory requirements, slow training and slow testing. It should be noted that input upsampling does not enrich the image details. Instead, it is needed because the higher convolutional layers respond very weakly to small objects. For example, a 32×32 object is mapped into a 4×4 patch of the "conv4-3" layer and a 2×2 patch of the "conv5-3" layer. This provides limited information for 7×7 ROI pooling.

To address this problem, we consider an efficient way to increase the resolution of feature maps. This consists of upsampling feature maps (instead of the input) using a deconvolution layer, as shown in Figure 2.4. This strategy is similar to that of [29], shown in Figure 2.2 (d), where

input rescaling is replaced by feature rescaling. In [29], a feature approximator is learned by least squares. In the CNN world, a better solution is to use a deconvolution layer, similar to that of [107]. Unlike input upsampling, feature upsampling does not incur in extra costs for memory and computation. Our experiments show that the addition of a deconvolution layer significantly boosts detection performance, especially for small objects. To the best of our knowledge, this is the first application of deconvolution to jointly improve the speed and accuracy of an object detector.

2.4.2 Context Embedding

Context has been shown useful for object detection [46, 16, 2] and segmentation [198]. Context information has been modeled by a recurrent neural network in [2] and acquired from multiple regions around the object location in [46, 16, 198]. In this work, we focus on context from multiple regions. As shown in Figure 2.4, features from an object (green cube) and a context (blue cube) region are stacked together immediately after ROI pooling. The context region is 1.5 times larger than the object region. An extra convolutional layer without padding is used to reduce the number of model parameters. It helps compress redundant context and object information, without loss of accuracy, and guarantees that the number of model parameters is approximately the same.

2.4.3 Implementation Details

Learning is initialized with the model generated by the first learning stage of the proposal network, described in Section 2.3.4. The learning rate is set to 0.0005, and reduced by a factor of 10 times after every 10,000 iterations. Learning stops after 25,000 iterations. The joint optimization of (2.6) is solved by back-propagation throughout the unified network. Bootstrapping is used and $\lambda = 1$. Following [50], the parameters of layers"conv1-1" to "conv2-2" are fixed

Table 2.1: Parameter configurations of the different models.

		det-8		det-16		det-32		det-64	ROI	FC
car	filter	5x5	7x7	5x5	7x7	5x5	7x7	5x5	7x7	4096
	anchor	40x40	56x56	80x80	112x112	160x160	224x224	320x320		
ped/cyc	filter	5x3	7x5	5x3	7x5	5x3	7x5	5x3	7x5	2048
	anchor	40x28	56x36	80x56	112x72	160x112	224x144	320x224		
caltech	filter	5x3	7x5	5x3	7x5	5x3	7x5	5x3	8x4	2048
	anchor	40x20	56x28	80x40	112x56	160x80	224x112	320x160		

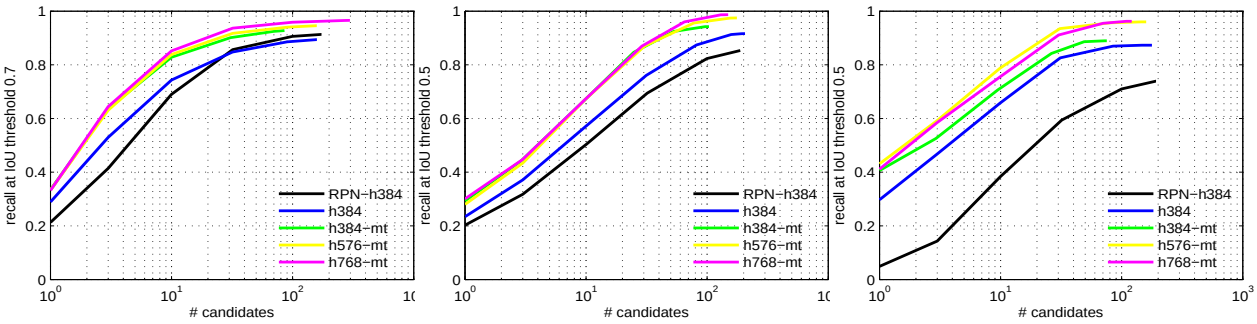
during learning, for faster training.

2.5 Experimental Evaluation

The performance of the MS-CNN detector was evaluated on the KITTI [43] and Caltech Pedestrian [32] benchmarks. These were chosen because, unlike VOC [36] and ImageNet [143], they contain many small objects. Typical image sizes are 1250×375 on KITTI and 640×480 on Caltech. KITTI contains three object classes: car, pedestrian and cyclist, and three levels of evaluation: easy, moderate and hard. The "moderate" level is the most commonly used. In total, 7,481 images are available for training/validation, and 7,518 for testing. Since no ground truth is available for the test set, we followed [16], splitting the trainval set into training and validation sets. In all ablation experiments, the training set was used for learning and the validation set for evaluation. Following [16], a model was trained for car detection and another for pedestrian/cyclist detection. One pedestrian model was learned on Caltech. The model configurations for original input size are shown in Table 2.1. The detector was implemented in C++ within the Caffe toolbox [76], and source code is available at https://github.com/zhaoweicai/mscnn. All times are reported for implementation on a single CPU core (2.40GHz) of an Intel Xeon E5-2630 server with 64GB of RAM. An NVIDIA Titan GPU was used for CNN computations.

Table 2.2: Detection recall of the various detection layers on KITTI validation set (car), as a function of object hight in pixels.

	det-8	det-16	det-32	det-64	combined
25≤height<50	0.9180	0.3071	0.0003	0	0.9360
50≤height<100	0.5934	0.9660	0.4252	0	0.9814
100≤height<200	0.0007	0.5997	0.9929	0.4582	0.9964
height≥200	0	0	0.9583	0.9792	0.9583
all scales	0.6486	0.5654	0.3149	0.0863	0.9611



Car Pedestrian Cyclist

Figure 2.5: Proposal recall on the KITTI validation set (moderate). "hXXX" refers to input images of height "XXX". "mt" indicates multi-task learning of proposal and detection sub-networks.

2.5.1 Proposal Evaluation

We start with an evaluation of the proposal network. Following [63], oracle recall is used as performance metric. For consistency with the KITTI setup, a ground truth is recalled if its best matched proposal has *IoU* higher than 70% for cars, and 50% for pedestrians and cyclists.

The roles of individual detection layers: Table 2.2 shows the detection accuracy of the various detection layers as a function of object height in pixels. As expected, each layer has highest accuracy for the objects that match its scale. While the individual recall across scales is low, the combination of all detectors achieves high recall for all object scales.

The effect of input size: Figure 2.5 shows that the proposal network is fairly robust to the size of input images for cars and pedestrians. For cyclist, performance increases between heights 384 and 576, but there are no gains beyond this. These results show that the network can

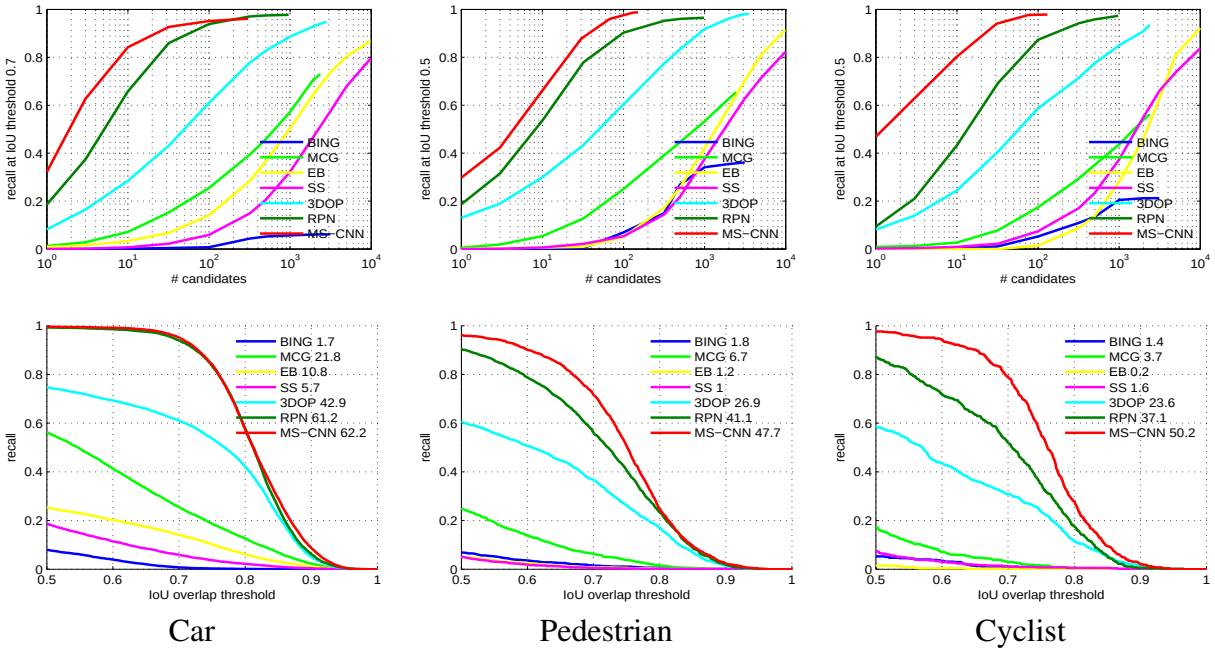


Figure 2.6: Proposal performance comparison on KITTI validation set (moderate). The first row is proposal recall curves and the second row is recall v.s. *IoU* for 100 proposals.

achieve good proposal generation performance without substantial input upsampling.

Detection sub-network improves proposal sub-network: [50] has shown that multi-task learning can benefit both bounding box regression and classification. On the other hand [140] showed that, even when features are shared between the two tasks, object detection does not improve object proposals too much. Figure 2.5 shows that, for the MS-CNN, detection can substantially benefit proposal generation, especially for pedestrians.

Comparison with the state-of-the-art: Figure 2.6 compares the proposal generation network to BING [17], Selective Search [158], EdgeBoxes [199], MCG [1], 3DOP [16] and RPN [140]. The top row of the figure shows that the MS-CNN achieves a recall about 98% with only 100 proposals. This should be compared to the $\sim$2,000 proposals required by 3DOP and the $\sim$10,000 proposals required by EdgeBoxbes. While it is not surprising that the proposed network outperforms unsupervised proposal methods, such as [158, 199, 1], its large gains over supervised methods [17, 16], that can even use 3D information, are significant. The closest performance is achieved by RPN (input upsampled twice), which has substantially weaker performance for

Table 2.3: Results on the KITTI validation set. "hXXX" indicates an input of height "XXX", "2x" deconvolution, "ctx" context encoding, and "c" dimensionality reduction convolution. In columns "Time" and "# params", entries before the "/" are for car model and after for pedestrian/cyclist model.

Model	Time	# params	Cars			Pedestrians		
			Easy	Mod	Hard	Easy	Mod	Hard
h384	0.11s/0.09s	471M/217M	90.90	80.63	68.94	73.70	68.37	60.72
h576	0.22s/0.19s	471M/217M	90.42	88.14	73.44	75.35	70.77	63.07
h768	0.41s/0.36s	471M/217M	89.84	88.88	75.78	76.38	72.26	64.08
h576-random	0.22s/0.19s	471M/217M	90.94	87.50	71.27	70.69	65.91	58.28
h576-mixture	0.22s/0.19s	471M/217M	90.33	88.12	72.90	75.09	70.49	62.43
h384-2x	0.12s/0.10s	471M/217M	90.55	87.93	71.90	76.01	69.53	61.57
h576-2x	0.23s/0.20s	471M/217M	94.08	89.12	75.54	77.74	72.49	64.43
h768-2x	0.43s/0.38s	471M/217M	90.96	88.83	75.19	76.33	72.71	64.31
h576-ctx	0.24s/0.20s	863M/357M	92.89	88.88	74.34	76.89	71.45	63.50
h576-ctx-c	0.22s/0.19s	297M/155M	90.49	89.13	74.85	76.82	72.13	64.14
proposal network (h576)	0.19s/0.18s	80M/78M	82.73	73.49	63.22	64.03	60.54	55.07

pedestrians and cyclists. When the input is not upsampled, RPN misses even more objects, as shown in Figure 2.5. It is worth mentioning that the MS-CNN generates high quality proposals (high overlap with the ground truth) without any edge detection or segmentation. This is evidence for the effectiveness of bounding box regression networks.

2.5.2 Object Detection Evaluation

In this section we evaluate object detection performance. Since the performance of the cyclist detector has large variance on the validation set, due to the low number of cyclist occurrences, only car and pedestrian detection are considered in the ablation experiments.

The effect of input upsampling: Table 2.3 shows that input upsampling can be a crucial factor for detection. A significant improvement is obtained by upsampling the inputs by 1.5$\sim$2 times, but we saw little gains beyond a factor of 2. This is smaller than the factor of 3.5 required by [16]. Larger factors lead to (exponentially) slower detectors and larger memory requirements.

Sampling strategy: Table 2.3 compares sampling strategies: random ("h576-random"), bootstrapping ("h576") and mixture ("h576-mixture"). For car, these three strategies are close to each other. For pedestrian, bootstrapping and mixture are close, but random is much worse. Note

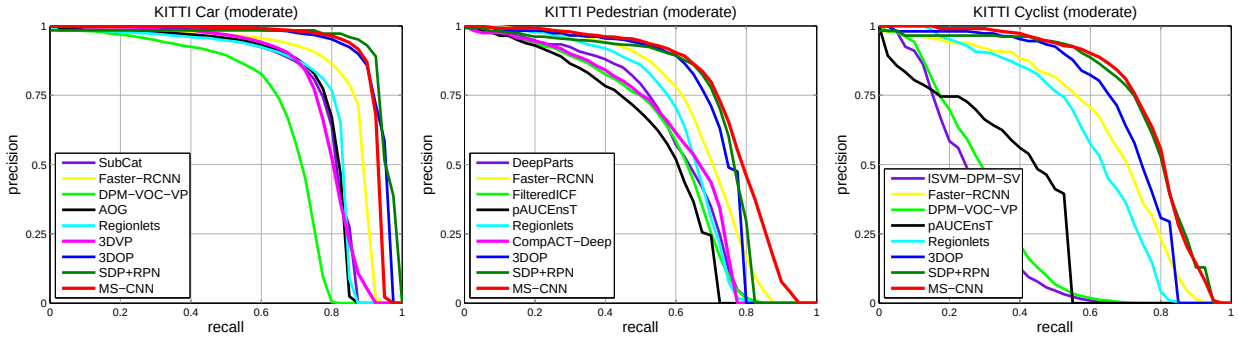


Figure 2.7: Comparison to the state-of-the-art on KITTI benchmark test set (moderate).

that random sampling has many more false positives than the other two.

CNN feature approximation: Three methods were attempted for learning the deconvolution layer for feature map approximation: 1) bilinearly interpolated weights; 2) weights initialized by bilinear interpolation and learned with back-propagation; 3) weights initialized with Gaussian noise and learned by back-propagation. We found the first method to work best, confirming the findings of [107, 173]. As shown in Table 2.3, the deconvoltion layer helps in most cases. The gains are larger for smaller input images, which tend to have smaller objects. Note that the feature map approximation adds trivial computation and no parameters.

Context embedding: Table 2.3 shows that there is a gain in encoding context. However, the number of model parameters almost doubles. The dimensionality reduction convolution layer significantly reduces this problem, without impairment of accuracy or speed.

Object detection by the proposal network: The proposal network can work as a detector, by switching the class-agnostic classification to class-specific. Table 2.3 shows that, although not as strong as the unified network, it achieves fairly good results, which are better than those of some detectors on the KITTI leaderboard[1].

Comparison to the state-of-the-art: The results of model "h768-ctx-c" were submitted to the KITTI leaderboard. A comparison to previous approaches is given in Table 2.4 and Figure 2.7. The MS-CNN set a new record for the detection of pedestrians and cyclists. The columns

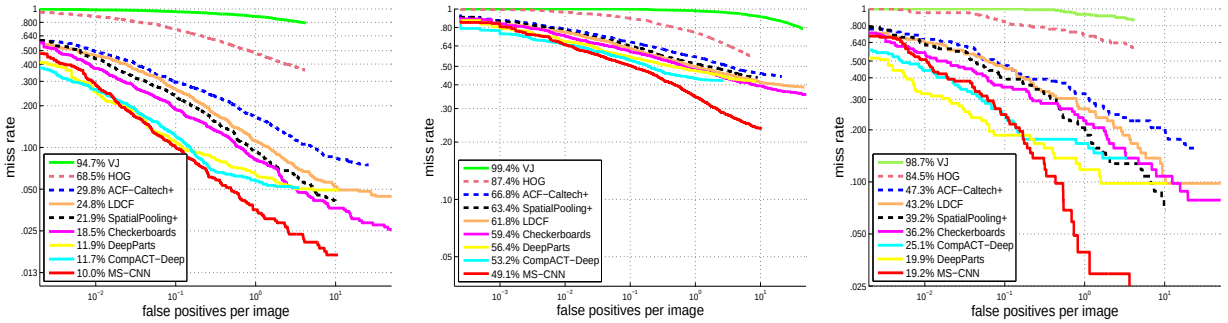
[1] http://www.cvlibs.net/datasets/kitti/

Table 2.4: Results on the KITTI benchmark test set (only published works shown).

Method	Time	Cars			Pedestrians			Cyclists		
		Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
LSVM-MDPM-sv [44]	10s	68.02	56.48	44.18	47.74	39.36	35.95	35.04	27.50	26.21
DPM-VOC-VP [132]	8s	74.95	64.71	48.76	59.48	44.86	40.37	42.43	31.08	28.23
SubCat [122]	0.7s	84.14	75.46	59.71	54.67	42.34	37.95	-	-	-
3DVP [169]	40s	87.46	75.77	65.38	-	-	-	-	-	-
AOG [89]	3s	84.80	75.94	60.70	-	-	-	-	-	-
Faster-RCNN [140]	2s	86.71	81.84	71.12	78.86	65.90	61.18	72.26	63.35	55.90
CompACT-Deep [11]	1s	-	-	-	70.69	58.74	52.71	-	-	-
DeepParts [154]	1s	-	-	-	70.49	58.67	52.78	-	-	-
FilteredICF [192]	2s	-	-	-	67.65	56.75	51.12	-	-	-
pAUCEnsT [126]	60s	-	-	-	65.26	54.49	48.60	51.62	38.03	33.38
Regionlets [164]	1s	84.75	76.45	59.70	73.14	61.15	55.21	70.41	58.72	51.83
3DOP [16]	3s	**93.04**	88.64	**79.10**	81.78	67.47	64.70	78.39	68.94	61.37
SDP+RPN [180]	0.4s	90.14	88.85	78.38	80.09	70.16	64.82	81.37	73.74	65.31
MS-CNN	0.4s	90.03	**89.02**	76.11	**83.92**	**73.70**	**68.31**	**84.06**	**75.46**	**66.07**

"Pedestrians-Mod" and "Cyclists-Mod" show substantial gains (6 and 7 points respectively) over 3DOP [16], and much better performance than the Faster-RCNN [140], Regionlets [164], etc. We also led a nontrivial margin over the very recent SDP+RPN [180], which used scale dependent pooling. In terms of speed, the network is fairly fast. For the largest input size, the MS-CNN detector is about 8 times faster than 3DOP. On the original images (1250×375) detection speed reaches 10 fps.

Pedestrian detection on Caltech: The MS-CNN detector was also evaluated on the Caltech pedestrian benchmark. The model "h720-ctx" was compared to methods such as DeepParts [154], CompACT-Deep [11], CheckerBoard [192], LDCF [121], ACF [29], and SpatialPooling [126] on three tasks: reasonable, medium and partial occlusion. As shown in Figure 2.8, the MS-CNN has state-of-the-art performance. Figure 2.8 (b) and (c) show that it performs very well for small and occluded objects, outperforming DeepParts [154], which explicitly addresses occlusion. Moreover, it misses a very small number of pedestrians, due to the accuracy of the proposal network. The speed is approximately 8 fps (15 fps) on upsampled 960×720 (original 640×480) Caltech images.



(a) reasonable (b) medium (c) partial occlusion

Figure 2.8: Comparison to the state-of-the-art on Caltech.

2.6 Conclusions

In this chapter, we have proposed a unified deep convolutional neural network, denoted the MS-CNN, for fast multi-scale object detection. The detection is preformed at various intermediate network layers, whose receptive fields match various object scales. This enables the detection of all object scales by feedforwarding a single input image through the network, which results in a very fast detector. CNN feature approximation was also explored, as an alternative to input upsampling. It was shown to result in significant savings in memory and computation. Overall, the MS-CNN detector achieves high detection rates at speeds of up to 15 fps.

2.7 Acknowledgements

Chapter 2 is, in full, based on the material as it appears in the publication of "A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection", Zhaowei Cai, Quanfu Fan, Rogerio Feris, and Nuno Vasconcelos, In *Proceedings of 14th European Conference on Computer Vision* (ECCV), 2016. The dissertation author was the primary investigator and author of this material.

Chapter 3

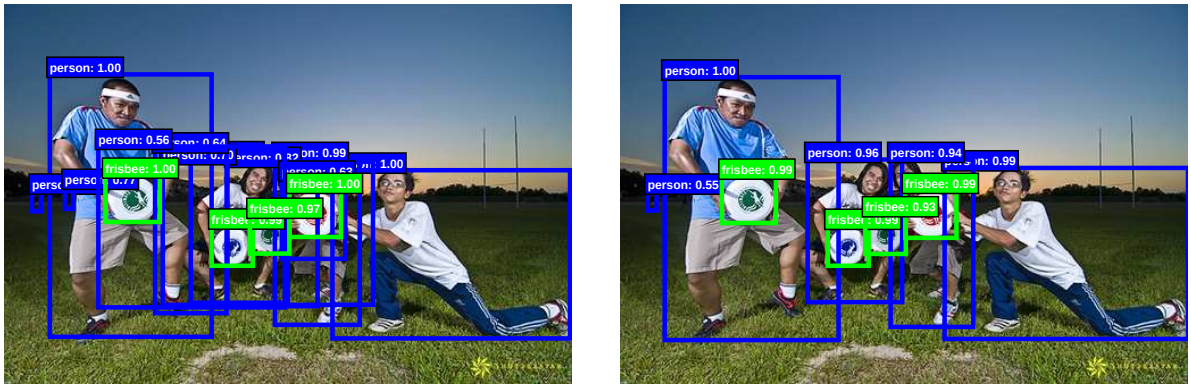
High-Quality Object Detection

3.1 Introduction

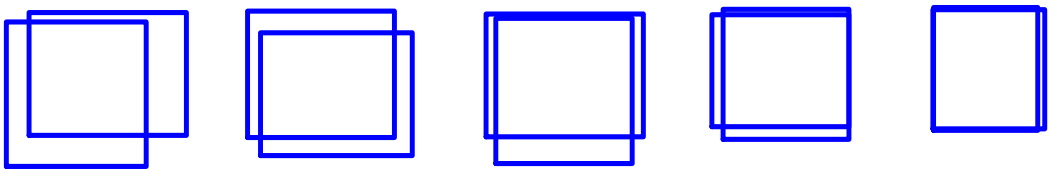
Object detection is a complex problem, requiring the solution of two tasks. First, the detector must solve the *recognition* problem, distinguishing foreground objects from background and assigning them the proper object class labels. Second, the detector must solve the *localization* problem, assigning accurate bounding boxes to different objects. An effective architecture for the solution of the two tasks, on which many of the recently proposed object detectors are based, is the two-stage R-CNN framework [51, 50, 140, 99]. This frames detection as a multi-task learning problem that combines classification, to solve the recognition problem, and bounding box regression, to solve localization.

Despite the success of this architecture, the two problems can be difficult to solve accurately. This is partly due to the fact that there are many "close" false positives, corresponding to "close but not correct" bounding boxes. An effective detector must find all true positives in an image, while suppressing these close false positives. This requirement makes detection more difficult than other classification problems, e.g. object recognition, where the difference between positives and negatives is not as fine-grained. In fact, the boundary between positives and negatives must be carefully defined. In the literature, this is done by thresholding the intersection over union (IoU) score between candidate and ground truth bounding boxes. While the threshold is typically set at the value of $u = 0.5$, this is a very loose requirement for positives. The resulting detectors frequently produce noisy bounding boxes, as shown in Figure 3.1 (a). Hypotheses that most humans would consider close false positives frequently pass the $IoU \geq 0.5$ test. While training examples assembled under the $u = 0.5$ criterion are rich and diverse, they make it difficult to train detectors that can effectively reject close false positives.

In this work, we define the *quality* of a detection hypothesis as its IoU with the ground truth, and the *quality of a detector* as the IoU threshold u used to train it. Some examples of hypotheses of increasing quality are shown in Figure 3.1 (c). The goal is to investigate the



(a) Detection of $u = 0.5$ (b) Detection of $u = 0.7$

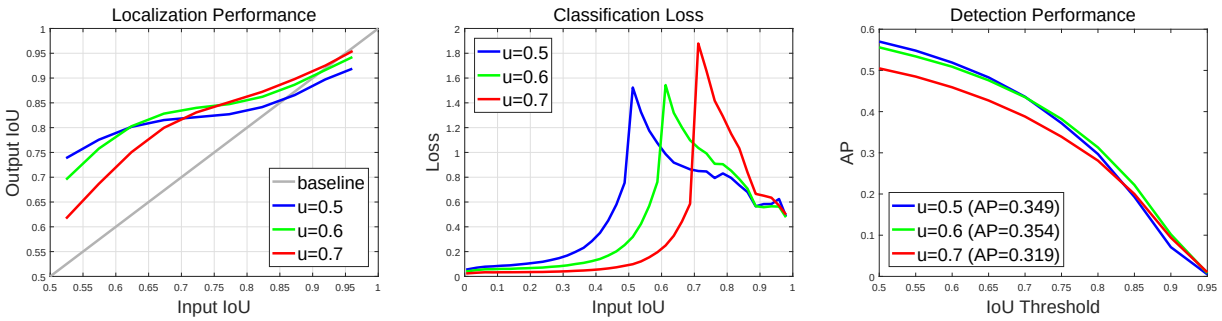


(c) Examples of increasing qualities

Figure 3.1: (a) and (b) detections by object detectors of increasing qualities, and (c) examples of increasing quality.

poorly researched problem of learning *high quality object detectors*. As shown in Figure 3.1 (b), these are detectors that produce few close false positives. The starting premise is that a single detector can only be optimal for a single quality level. This is known in the cost-sensitive learning literature [35, 113], where the optimization of different points of the receiver operating characteristic (ROC) requires different loss functions. The main difference is that we consider the optimization for a given IoU threshold, rather than false positive rate.

Some evidence in support of this premise is given in Figure 3.2, which presents the bounding box localization performance, classification loss and detection performance, respectively, of three detectors trained with IoU thresholds of $u = 0.5, 0.6, 0.7$. Localization and classification are evaluated as a function of the detection hypothesis IoU. Detection is evaluated as a function of the IoU threshold, as in COCO [98]. Figure 3.2 (a) shows that the three bounding box regressors tend to achieve the best performance for examples of IoU in the vicinity of the threshold used for



(a) Regressor (b) Classifier (c) Detector

Figure 3.2: Bounding box localization, classification loss and detection performance of object detectors of increasing IoU threshold u.

detector training. Figure 3.2 (c) shows a similar effect for detection, up to some overfitting for the highest thresholds. The detector trained with $u = 0.5$ outperforms the detector trained with $u = 0.6$ for low IoUs, underperforming it at higher IoUs. In general, a detector optimized for a single IoU value is not optimal for other values. This is also confirmed by the classification loss, shown in Figure 3.2 (b), whose peaks are near the thresholds used for detector training. In general, the threshold determines the classification boundary where the classifier is most discriminative, i.e. has largest margin [19, 40].

The observations above suggest that high quality detection requires a close *match* between the quality of the detector and that of the detection hypotheses. The detector will only achieve high quality if presented with high quality proposals. This, however, cannot be guaranteed by simply increasing the threshold u during training. On the contrary, as seen for the detector of $u = 0.7$ in Figure 3.2 (c), forcing a high value of u usually degrades detection performance. We refer to this problem, i.e. that training a detector with higher threshold leads to poorer performance, as the *paradox of high-quality detection*. This problem has two causes. First, object proposal mechanisms tend to produce hypotheses distributions heavily imbalanced towards low quality. In result, the use of larger IoU thresholds during training exponentially reduces the number of positive training examples. This is particularly problematic for neural networks, which are very example intensive, making the "high u" training strategy very prone to overfitting. Second, there

is a mismatch between the quality of the detector and that of the hypotheses available at inference time. Since, as shown in Figure 3.2, high quality detectors are only optimal for high quality hypotheses, detection performance can degrade substantially for hypotheses of lower quality.

In this chapter, we propose a new detector architecture, denoted as *Cascade R-CNN,* that addresses these problems, to enable high quality object detection. The new architecture is a multi-stage extension of the R-CNN, where detector stages deeper into the cascade are sequentially more selective against close false positives. As is usual for classifier cascades [161, 144], the cascade of R-CNN stages is trained sequentially, using the output of one stage to train the next. This leverages the observation that the output IoU of a bounding box regressor is almost always better than its input IoU, as can be seen in Figure 3.2 (a), where nearly all plots are above the gray line. In result, the output of a detector trained with a certain IoU threshold is a good hypothesis distribution to train the detector of the next higher IoU threshold. This has some similarity to *boostrapping* methods commonly used to assemble datasets for object detection [161, 38]. The main difference is that the resampling performed by the Cascade R-CNN does not aim to mine hard negatives. Instead, by adjusting bounding boxes, each stage aims to find a good set of close false positives for training the next stage. The main outcome of this resampling is that the quality of the detection hypotheses increases *gradually*, from one stage to the next. In result, the sequence of detectors addresses the two problems underlying the paradox of high-quality detection. First, because the resampling operation guarantees the availability of a *large number* of examples for the training of all detectors in the sequence, it is possible to train detectors of high IoU without overfitting. Second, the use of the same cascade procedure at inference time produces a set of hypotheses of progressively higher quality, well *matched* to the increasing quality of the detector stages. This enables higher detection accuracies, as suggested by Figure 3.2.

The Cascade R-CNN is quite simple to implement and trained end-to-end. Our results show that a vanilla implementation, without any bells and whistles, surpasses almost all previous state-of-the-art *single-model* detectors, on the challenging COCO detection task [98], especially

under the stricter evaluation metrics. In addition, the Cascade R-CNN can be built with any two-stage object detector based on the R-CNN framework. We have observed consistent gains (of $2\sim4$ points, and more under stricter localization metrics), at a marginal increase in computation. This gain is independent of the strength of the baseline object detectors, for all the models we have tested. We thus believe that this simple and effective detection architecture can be of interest for many object detection research efforts.

A preliminary version of this manuscript was previously published in [12]. After the original publication, the Cascade R-CNN has been successfully reproduced within many different codebases, including the popular Detectron [49], PyTorch[1], and TensorFlow[2], showing consistent and reliable improvements independently of implementation codebase. In this expanded version, we have extended the Cascade R-CNN to instance segmentation, by adding a mask head to the cascade, denoted as *Cascade Mask R-CNN*. This is shown to achieve non-trivial improvements over the popular Mask R-CNN [58]. A new and more extensive evaluation is also presented, showing that the Cascade R-CNN is compatible with many complementary enhancements proposed in the detection and instance segmentation literatures, some of which were introduced after [12], e.g. GroupNorm [167]. Finally, we further present the results of a larger set of experiments, performed on various popular generic/specific object detection datasets, including PASCAL VOC [36], KITTI [43], CityPerson [193] and WiderFace [182]. These experiments demonstrate that the paradox of high quality object detection applies to all these tasks, and that the Cascade R-CNN enables more effective high quality detection than previously available methods. Due to these properties, as well as its generality and flexibility, the Cascade R-CNN has recently been adopted by the winning teams of the COCO 2018 instance segmentation challenge[3], the OpenImage 2018 challenge[4], and the Wider Challenge 2018[5]. To facilitate future research, we

[1] https://github.com/open-mmlab/mmdetection
[2] https://github.com/tensorpack/tensorpack
[3] http://cocodataset.org/#detection-leaderboard
[4] https://storage.googleapis.com/openimages/web/challenge.html
[5] http://wider-challenge.org/

have released the code on two codebases, https://github.com/zhaoweicai/cascade-rcnn (Caffe [76]) and https://github.com/zhaoweicai/Detectron-Cascade-RCNN (Detectron [49]) to facilitate future research.

3.2 Related Work

Due to the success of the R-CNN [51] detector, which combines a proposal detector and a region-wise classifier, this two-stage architecture has become predominant in the recent past. To reduce redundant CNN computations, the SPP-Net [59] and Fast R-CNN [50] introduced the idea of region-wise feature extraction, enabling the sharing of the bulk of feature computations by object instances. The Faster R-CNN [140] then achieved further speeds-up by introducing a region proposal network (RPN), becoming the cornerstone of modern object detection. Later, some works extended this detector to address various problems of detail. For example, the R-FCN [24] proposed efficient region-wise full convolutions to avoid the heavy CNN computations of the Faster R-CNN; and the Mask R-CNN [58] added a network head that computes object masks to support instance segmentation. Some more recent works have focused on normalizing feature statistics [131, 167], modeling relations between instances [65], non maximum suppression (NMS) [7], and other aspects [149, 103].

Scale invariance, an important requisite for effective object detection, has also received substantial attention in the literature [10, 99, 151]. While natural images contain objects at various scales, the fixed receptive field size of the filters implemented by the RPN [140] makes it prone to scale mismatches. To overcome this, the MS-CNN [10] introduced a multi-scale object proposal network, by generating outputs at multiple layers. This leverages the different receptive field sizes of the different layers to produce a set of scale-specific proposal generators, which is then combined into a strong multi-scale generator. Similarly, the FPN [99] detects high-recall proposals at multiple output layers, with recourse to a scale-invariant feature representation

by adding a top-down connection across feature maps of different network depths. Both the MS-CNN and FPN rely on a feature pyramid representation for multi-scale object detection. SNIP [151], on the other hand, recently revisited image pyramid in modern object detection. It normalizes the gradients from different object scales during training, such that the whole detector is scale-specific. Scale-invariant detection is achieved by using an image pyramid at inference.

One-stage object detection architectures have also become popular for their computational efficiency. YOLO [137] outputs very sparse detection results and enables real-time object detection, by forwarding the input image once through an efficient backbone network. SSD [104] detects objects in a way similar to the RPN [140], but uses multiple feature maps at different resolutions to cover objects at various scales. The main limitation of these detectors is that their accuracy is typically below that of two-stage detectors. The RetinaNet [100] detector was proposed to address the extreme foreground-background class imbalance of dense object detection, achieving results comparable to two-stage detectors. Recently, CornerNet [87] proposed to detect an object bounding box as a pair of keypoints, abandoning the widely used concept of anchors first introduced by the Faster R-CNN. This detector has achieved very good performance with the help of some training and testing enhancements. RefineDet [194] added an anchor refinement module to the single-shot SSD [104], to improve localization accuracy. This is somewhat similar to the cascaded localization implemented by the proposed Cascade R-CNN, but ignores the problem of high-quality detection.

Some explorations in multi-stage object detection have also been proposed. The multi-region detector of [46] introduced *iterative bounding box regression*, where a R-CNN is applied several times to produce successively more accurate bounding boxes. [179, 48, 47] used a multi-stage procedure to generate accurate proposals, which are forwarded to an accurate model (e.g. Fast R-CNN). [186, 118] proposed an alternative procedure to localize objects sequentially. While this is similar in spirit to the Cascade-RCNN, these methods use the *same* regressor iteratively for accurate localization. On the other hand, [91, 123] embedded the classic cascade

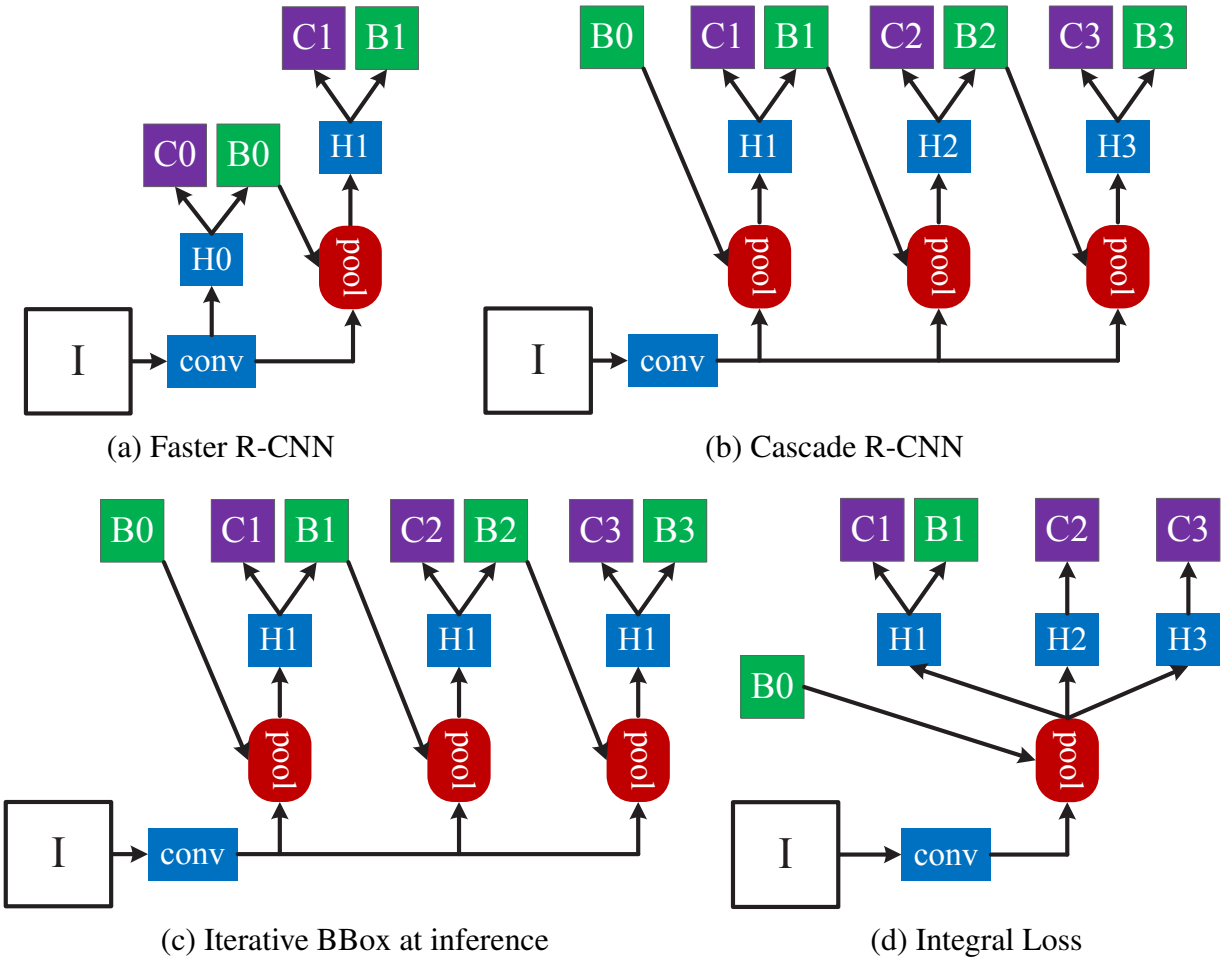


Figure 3.3: The architectures of different frameworks. "I" is input image, "conv" backbone convolutions, "pool" region-wise feature extraction, "H" network head, "B" bounding box, and "C" classification. "B0" is proposals in all architectures.

architecture of [161] in an object detection network. Finally, [23] iterated between the detection and segmentation tasks, to achieve improved instance segmentation.

Upon publication of the conference version of this manuscript, several works have pursued the idea behind Cascade R-CNN [77, 105, 166]. [105, 166] applied it to single-shot object detectors, showing nontrivial improvements for high quality single-shot detection, for general objects and pedestrians, respectively. The IoU-Net [77] explored in greater detail high-quality localization, achieving some gains over the Cascade R-CNN by cascading more bounding box regression steps. [57] showed it is possible to achieve state-of-the-art object detectors without

ImageNet pretraining, with a help of the Cascade R-CNN. These works show that the Cascade R-CNN idea is robust and applicable to various object detection architectures. This suggests that it should continue to be useful despite future advances in object detection.

3.3 Review of High-Quality Object Detection

In this section, we discuss the challenges of high quality object detection.

3.3.1 Object Detection

While the ideas proposed in this work can be applied to various detector architectures, we focus on the popular two-stage architecture of the Faster R-CNN [140], shown in Figure 3.3 (a). The first stage is a proposal sub-network, in which the entire image is processed by a *backbone* network, e.g. ResNet [61], and a proposal head ("H0") is applied to produce preliminary detection hypotheses, known as object proposals. In the second stage, these hypotheses are processed by a region-of-interest detection sub-network ("H1"), denoted as a *detection head*. A final classification score ("C") and a bounding box ("B") are assigned per hypothesis. The entire detector is learned end-to-end, using a multi-task loss with bounding box regression and classification components.

Bounding Box Regression: A bounding box $\mathbf{b} = (b_x, b_y, b_w, b_h)$ contains the four coordinates of an image patch $\mathbf{x}$. Bounding box regression aims to regress a candidate bounding box $\mathbf{b}$ into a target bounding box $\mathbf{g}$, using a regressor $f(\mathbf{x}, \mathbf{b})$. This is learned from a training set $(\mathbf{g}_i, \mathbf{b}_i)$, by minimizing the risk

$$\mathcal{R}_{loc}[f] = \sum_i L_{loc}(f(\mathbf{x}_i, \mathbf{b}_i), \mathbf{g}_i). \tag{3.1}$$

As in Fast R-CNN [50],

$$L_{loc}(\mathbf{a}, \mathbf{b}) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(a_i - b_i) \tag{3.2}$$

where

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2, & |x| < 1 \\ |x| - 0.5, & \text{otherwise,} \end{cases} \tag{3.3}$$

is the smooth L_1 loss function. To encourage invariance to scale and location, $smooth_{L_1}$ operates on the distance vector $\Delta = (\delta_x, \delta_y, \delta_w, \delta_h)$ defined by

$$\begin{aligned} \delta_x = (g_x - b_x)/b_w, \quad \delta_y = (g_y - b_y)/b_h \\ \delta_w = \log(g_w/b_w), \quad \delta_h = \log(g_h/b_h). \end{aligned} \tag{3.4}$$

Since bounding box regression usually performs minor adjustments on $\mathbf{b}$, the numerical values of (3.4) can be very small. This usually makes the regression loss much smaller than the classification loss. To improve the effectiveness of multi-task learning, Δ is normalized by its mean and variance, e.g. δ_x is replaced by

$$\delta'_x = \frac{\delta_x - \mu_x}{\sigma_x}. \tag{3.5}$$

This is widely used in the literature [140, 10, 24, 99, 58].

Classification: The classifier is a function $h(\mathbf{x})$ that assigns an image patch $\mathbf{x}$ to one of $M+1$ classes, where class 0 contains background and the remaining classes the objects to detect. $h(\mathbf{x})$ is a $M+1$-dimensional estimate of the posterior distribution over classes, i.e. $h_k(\mathbf{x}) = p(y = k|\mathbf{x})$, where y is the class label. Given a training set $(\mathbf{x}_i, y_i)$, it is learned by minimizing the classification risk

$$\mathcal{R}_{cls}[h] = \sum_i L_{cls}(h(\mathbf{x}_i), y_i), \tag{3.6}$$

where

$$L_{cls}(h(\mathbf{x}), y) = -\log h_y(\mathbf{x}) \tag{3.7}$$

is the cross-entropy loss.

3.3.2 Detection Quality

Consider a ground truth object of bounding box $\mathbf{g}$ associated with class label y, and a detection hypothesis $\mathbf{x}$ of bounding box $\mathbf{b}$. Since a $\mathbf{b}$ usually includes an object and some amount of background, it can be difficult to determine if a detection is correct or not. This is usually addressed by the intersection over union (IoU) metric

$$IoU(\mathbf{b},\mathbf{g}) = \frac{\mathbf{b} \cap \mathbf{g}}{\mathbf{b} \cup \mathbf{g}}. \tag{3.8}$$

If the IoU is above a threshold u, the patch is considered an example of the class of the object of bounding box $\mathbf{g}$ and denoted "positive". Thus, the class label of a hypothesis $\mathbf{x}$ is a function of u,

$$y_u = \begin{cases} y, & IoU(\mathbf{b},\mathbf{g}) \geq u \\ 0, & \text{otherwise.} \end{cases} \tag{3.9}$$

If the IoU does not exceed the threshold for any object, $\mathbf{x}$ is assigned to the background and denoted "negative".

Although there is no need to define positive/neagtive examples for the bounding box regression task, an IoU threshold u is also required to select the set of samples

$$\mathcal{G} = \{(\mathbf{g}_i, \mathbf{b}_i) | IoU(\mathbf{b}_i, \mathbf{g}_i) \geq u\} \tag{3.10}$$

used to train the regressor. While the IoU thresholds used for the two tasks do not have to be identical, this is usual in practice. Hence, the IoU threshold u defines the *quality* of a detector. Large thresholds encourage detected bounding boxes to be tightly aligned with their ground truth counterparts. Small thresholds reward detectors that produce loose bounding boxes, of small overlap with the ground truth.

A main challenge of object detection is that, no matter the choice of threshold, the

detection setting is highly adversarial. When u is high, positives contain less background but it is difficult to assemble large positive training sets. When u is low, richer and more diverse positive training sets are possible, but the trained detector has little incentive to reject close false positives. In general, it is very difficult to guarantee that a single classifier performs uniformly well over all IoU levels. At inference, since the majority of the hypotheses produced by a proposal detector, e.g. RPN [140] or selective search [156], have low quality, the detector must be more discriminant for lower quality hypotheses. A standard compromise between these conflicting requirements is to settle on $u = 0.5$, which is used in almost *all* modern object detectors. This, however, is a relatively low threshold, leading to low quality detections that most humans consider close false positives, as shown in Figure 3.1 (a).

3.3.3 Challenges to High Quality Detection

Despite the significant progress in object detection of the past few years, few works attempted to address high quality detection. This is mainly due to the following reasons.

First, evaluation metrics have historically placed greater emphasis on the low quality detection regime. For performance evaluation, an IoU threshold u is used to determine whether a detection is a success ($IoU(\mathbf{b},\mathbf{g}) \geq u$) or failure ($IoU(\mathbf{b},\mathbf{g}) < u$). Many object detection datasets, including PASCAL VOC [36], ImageNet [143], Caltech Pedestrian[32], etc., use $u = 0.5$. This is partly because these datasets were established a while ago, when object detection performance was far from what it is today. However, this loose evaluation standard is adopted even by relatively recent datasets, such as WiderFace [182], or CityPersons [193]. This is one of the main reasons why performance has saturated for many of these datasets. Others, such as COCO [98] or KITTI [43] use stricter evaluation metrics: average precision at $u = 0.7$ for car in KITTI, and mean average precision across $u = [0.5 : 0.05 : 0.95]$ in COCO. While recent works have focused on these less saturated datasets, most detectors are still designed with the loose IoU threshold of $u = 0.5$, associated with the low-quality detection regime. In this work, we show that there is

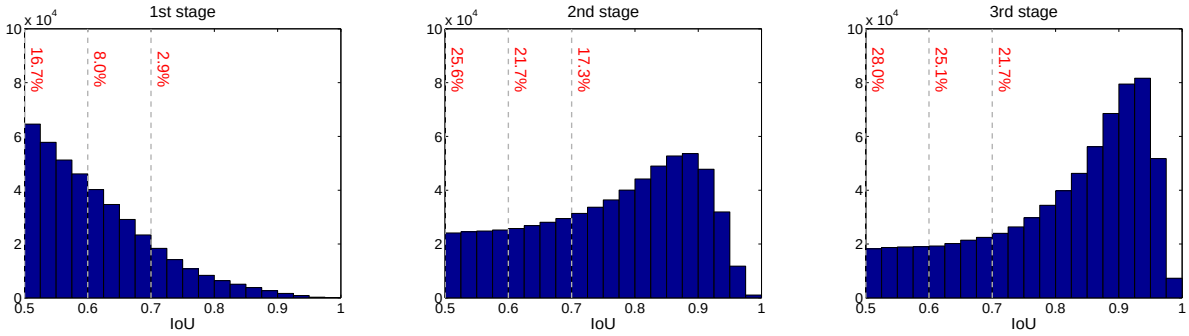


Figure 3.4: IoU histograms of training samples of each cascade stage. The distribution of the 1st stage is the RPN output. Shown in red are the percentage of positives for the corresponding IoU threshold.

plenty of room for improvement when a stricter evaluation metric, e.g. $u \geq 0.75$, is used and that it is possible to achieve significant improvements by designing detectors specifically for the high quality regime.

Second, the design of high quality object detectors is not a trivial generalization of existing approaches, due to the paradox of high quality detection. To beat the paradox, it is necessary to match the qualities of the hypotheses generator and the object detector. In the literature, there have been efforts to increase the quality of hypotheses, e.g. by iterative bounding box regression [48, 47] or better RPN design [10, 99], and some efforts to increase the quality of the object detector, e.g. by using the integral loss on a set of IoU thresholds [187]. These attempts fail to guarantee high quality detection because they consider only one of the goals, missing the fact that the qualities of both tasks need to be increased *simultaneously*. On one hand, raising the quality of the hypotheses has little benefit if the detector remains of low quality, because the latter is not trained to discriminate high quality from low quality hypotheses. On the other, if only the detector quality is increased, there are too few high quality hypotheses for it to classify, leading to no detection improvement. In fact, because, as shown in Figure 3.4 (left), the set of positive samples decreases quickly with u, a high u detector is prone to overfitting. Hence, a high u detector can easily overfit and perform worse than a low u detector, as shown in Figure 3.2 (c).

3.4 Cascade R-CNN

In this section we introduce the Cascade R-CNN detector.

3.4.1 Architecture

The architecture of the Cascade R-CNN is shown in Figure 3.3 (b). It is a multi-stage extension of the Faster R-CNN architecture of Figure 3.3 (a). In this work, we focus on the the detection sub-network, simply adopting the RPN [140] of Figure 3.3 (a) for proposal detection. However, the Cascade R-CNN is not limited to this proposal mechanism, other choices should be possible. As discussed in the section above, the goal is to increase the quality of hypotheses and detector *simultaneously*, to enable high quality object detection. This is achieved with a combination of cascaded bounding box regression and cascaded detection.

3.4.2 Cascaded Bounding Box Regression

High quality hypotheses can be easily produced during training, where ground truth bounding boxes are available, e.g. by sampling around the ground truth. The difficulty is to produce high quality proposals at inference, when ground truth is unavailable. This problem is addressed with resort to cascaded bounding box regression.

As shown in Figure 3.2 (a), a single regressor cannot usually perform uniformly well over all quality levels. However, as is commonly done for pose regression [31] or face alignment [13, 174, 175], the regression task can be decomposed into a sequence of simpler steps. In the Cascade R-CNN detector, the idea is implemented with a cascaded regressor with the architecture of Figure 3.3 (b). This consists of a cascade of *specialized* regressors

$$f(\mathbf{x}, \mathbf{b}) = f_T \circ f_{T-1} \circ \cdots \circ f_1(\mathbf{x}, \mathbf{b}), \tag{3.11}$$

where T is the total number of cascade stages. The key point is that each regressor f_t is optimized for the bounding box distribution $\{\mathbf{b}^t\}$ generated by the previous regressor, rather than the initial distribution $\{\mathbf{b}^1\}$. In this way, the hypotheses are improved *progressively.*

This is illustrated in Figure 3.5, which presents the distribution of the regression distance vector $\Delta = (\delta_x, \delta_y, \delta_w, \delta_h)$ at different cascade stages. Note that most hypotheses become closer to the ground truth as they progress through the cascade. There are also some hypotheses that fail to meet the stricter IoU criteria of the later cascade stages. These are declared outliers and eliminated. It should be noted that, as discussed in Section 3.3.1, Δ needs be mean/variance normalized, as in (3.5), for effective multi-task learning. The mean and variance statistics computed after this outlier removal step are used to normalize Δ at each cascade stage. Our experiments show that this implementation of cascaded bounding box regression generates hypotheses of very high quality at both training and inference.

3.4.3 Cascaded Detection

As shown in the left of Figure 3.4, the initial hypotheses distribution produced by the RPN is heavily tilted towards low quality. For example, only 2.9% of examples are positive for an IoU threshold $u = 0.7$. This makes it difficult to train a high quality detector. The Cascade R-CNN addresses the problem by using cascade regression as a *resampling mechanism*. This is inspired by Figure 3.2 (a), where nearly all curves are above the diagonal gray line, showing that a bounding box regressor trained for a certain u tends to produce bounding boxes of *higher* IoU. Hence, starting from examples $\{(\mathbf{x}_i, \mathbf{b}_i)\}$, cascade regression successively resamples an example distribution $\{(\mathbf{x}'_i, \mathbf{b}'_i)\}$ of higher IoU. This enables the sets of positive examples of the successive stages to keep a roughly *constant* size, even when the detector quality u is *increased*. Figure 3.4 illustrates this property, showing how the example distribution tilts more heavily towards high quality examples after each resampling step.

At each stage t, the R-CNN head includes a classifier h_t and a regressor f_t optimized for

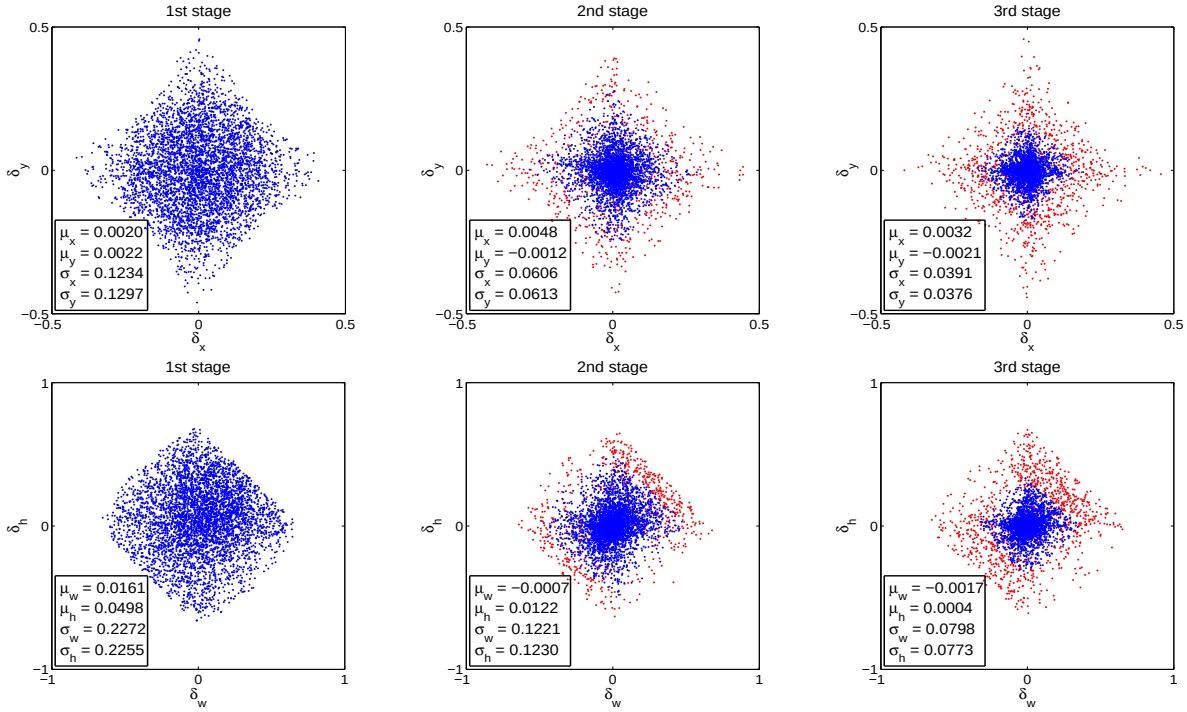


Figure 3.5: Distribution of the distance vector Δ of (3.4) (without normalization) at different cascade stages. Top: plot of (δ_x, δ_y). Bottom: plot of (δ_w, δ_h). Red dots are outliers for the increasing IoU thresholds of later stages, and the statistics shown are obtained after outlier removal.

the corresponding IoU threshold u^t, where $u^t > u^{t-1}$. These are learned with loss

$$L(\mathbf{x}^t, g) = L_{cls}(h_t(\mathbf{x}^t), y^t) + \lambda[y^t \geq 1] L_{loc}(f_t(\mathbf{x}^t, \mathbf{b}^t), \mathbf{g}), \tag{3.12}$$

where $\mathbf{b}^t = f_{t-1}(\mathbf{x}^{t-1}, \mathbf{b}^{t-1})$, $\mathbf{g}$ is the ground truth object for $\mathbf{x}^t$, $\lambda = 1$ the trade-off coefficient, y^t is the label of $\mathbf{x}^t$ under the u^t criterion, according to (3.9), $[\cdot]$ is the indicator function. Note that the use of $[\cdot]$ implies that the IoU threshold u of bounding box regression is identical to that used for classification. This cascade learning has three important consequences for detector training. First, the potential for overfitting at large IoU thresholds u is reduced, since positive examples become plentiful at all stages (see Figure 3.4). Second, detectors of deeper stages are optimal for higher IoU thresholds. Third, because some outliers are removed as the IoU threshold increases (see Figure 3.5), the learning effectiveness of bounding box regression increases in

the later stages. This simultaneous improvement of hypotheses and detector quality enables the Cascade R-CNN to beat the paradox of high quality detection. At inference, the same cascade is applied. The quality of the hypotheses is improved sequentially, and higher quality detectors are only required to operate on higher quality hypotheses, for which they are optimal. This enables the high quality object detection results of Figure 3.1 (b), as suggested by Figure 3.2.

3.4.4 Differences from Previous Works

The Cascade R-CNN has similarities to previous works using *iterative bounding box regression* and *integral loss* for detection. There are, however, important differences.

Iterative Bounding Box Regression: Some works [46, 47, 61] have previously argued that the use of a single bounding box regressor f is insufficient for accurate localization. These methods apply f iteratively, as a post-processing step

$$f'(\mathbf{x}, \mathbf{b}) = f \circ f \circ \cdots \circ f(\mathbf{x}, \mathbf{b}), \tag{3.13}$$

that refines a bounding box $\mathbf{b}$. This is called *iterative bounding box regression* and denoted as *iterative BBox*. It can be implemented with the inference architecture of Figure 3.3 (c) where all heads are *identical*. Note that this is only for inference, as training is identical to that of a two-stage object detector, e.g. the Faster R-CNN of Figure 3.3 (a) with $u = 0.5$. This approaches ignores two problems. First, as shown in Figure 3.2, a regressor f trained at $u = 0.5$ is suboptimal for hypotheses of higher IoUs. It actually *degrades* bounding box accuracy for IoUs larger than 0.85. Second, as shown in Figure 3.5, the distribution of bounding boxes changes significantly after each iteration. While the regressor is optimal for the initial distribution it can be quite suboptimal after that. Due to these problems, *iterative BBox* requires a fair amount of human engineering, in the form of proposal accumulation, box voting, etc, and has somewhat unreliable gains [46, 47, 61]. Usually, there is no benefit beyond applying f twice.

The Cascade R-CNN differs from *iterative BBox* in several ways. First, while *iterative BBox* is a post-processing procedure used to improve bounding boxes, the Cascade R-CNN uses cascade regression as a *resampling* mechanism that changes the distribution of hypotheses processed by the different stages. Second, because cascade regression is used at both training and inference, there is no discrepancy between training and inference distributions. Third, the multiple specialized regressors $\{f_T, f_{T-1}, \cdots, f_1\}$ are optimal for the *resampled distributions* of the different stages. This is unlike the single f of (3.13), which is only optimal for the initial distribution. Our experiments show that the Cascade R-CNN enables more precise localization than that possible with *iterative BBox*, and requires no human engineering.

Integral Loss: [187] proposed an ensemble of classifiers with the architecture of Figure 3.3 (d) and trained with the integral loss. This is a loss

$$L_{cls}(h(\mathbf{x}), y) = \sum_{u \in U} L_{cls}(h_u(\mathbf{x}), y_u) \tag{3.14}$$

that targets various quality levels, defined by a set of IoU thresholds $U = \{0.5, 0.55, \cdots, 0.75\}$, chosen to fit the evaluation metric of the COCO challenge.

The Cascade R-CNN differs from this detector in several ways. First, (3.14) fails to address the problem that the various loss terms operate on different numbers of positives. As shown on Figure 3.4 (left), the set of positive samples decreases quickly with u. This is particularly problematic because it makes the high quality classifiers very prone to overfitting. On the other hand, as shown in Figure 3.4, the resampling of the Cascade R-CNN produces a nearly constant number of positive examples as the IoU threshold u increases. Second, at inference, the high quality classifiers are required to process proposals of overwhelming low quality, for which they are not optimal. This is unlike the higher quality detectors of the Cascade R-CNN, which are only required to operate on higher quality hypotheses. Third, the integral loss is designed to fit the COCO metrics and, by definition, the classifiers are ensembled at inference. The Cascade

R-CNN aims to achieve high quality detection, and the high quality detector itself in the last stage can obtain the state-of-the-art detection performance. Due to all this, the *integral loss* detector of Figure 3.3 (d) usually fails to outperform the vanilla detector of Figure 3.3 (a), for most quality levels. This is unlike the Cascade R-CNN, which can have significant improvements.

3.5 Instance Segmentation

Instance segmentation has become popular in the recent past [23, 58, 103]. It aims to predict pixel-level segmentation for each instance, in addition to determining its object class. This is more difficult than object detection, which only predicts a bounding box (plus class) per instance. In general, instance segmentation is implemented in addition to object detection, and a stronger object detector usually leads to improved instance segmentation. The most popular instance segmentation method is arguably the Mask R-CNN [58]. Like the Cascade R-CNN, it is a variant on the two-stage detector. In this section, we extend the Cascade R-CNN architecture to the instance segmentation task, by adding a segmentation branch similar to that of the Mask R-CNN.

3.5.1 Mask R-CNN

The Mask R-CNN [58] extends the Faster R-CNN by adding a segmentation branch in parallel to the existing detection branch during training. It has the architecture of Figure 3.6 (a). The training instances are the positive examples also used to train the detection task. At inference, object detections are complemented with segmentation masks, for all detected objects.

3.5.2 Cascade Mask R-CNN

In the Mask R-CNN, the segmentation branch is inserted in parallel to the detection branch. However, the Cascade R-CNN has multiple detection branches. This raises the questions

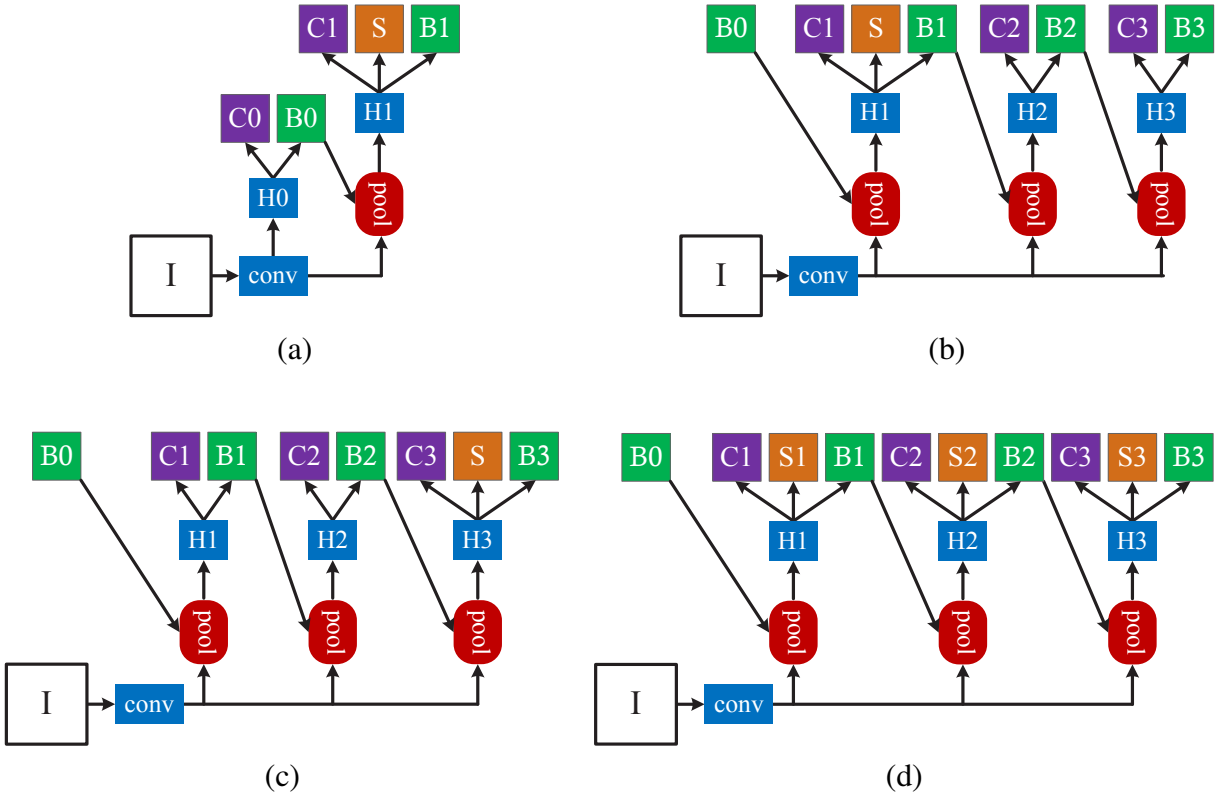


Figure 3.6: Architectures of the Mask R-CNN (a) and three Cascade Mask R-CNN strategies for instance segmentation (b)-(d). Beyond the definitions of Figure 3.3, "S" denotes a segmentation branch. Note that segmentations branches do not necessarily share heads with the detection branch.

of 1) where to add the segmentation branch and 2) how many segmentation branches to add. We consider three strategies for mask prediction in the Cascade R-CNN. The first two strategies address the first question, adding a single mask prediction head at either the first or last stage of the Cascade R-CNN, as shown in Figure 3.6 (b) and (c), respectively. Since the instances used to train the segmentation branch are the positives of the detection branch, their number varies in these two strategies. As shown in Figure 3.4, placing the segmentation head later on the cascade leads to more examples. However, because segmentation is a pixel-wise operation, a large number of highly overlapping instances is not necessarily as helpful as for object detection, which is a patch-based operation. The third strategy addresses the second question, adding a segmentation branch to each cascade stage, as shown in Figure 3.6 (d). This maximizes the diversity of samples

used to learn the mask prediction task.

At inference time, all three strategies predict the segmentation masks on the patches produced by the final object detection stage, irrespective of the cascade stage on which the segmentation mask is implemented and how many segmentation branches there are. The final mask prediction is obtained from the single segmentation branch for the architectures of Figure 3.6 (b) and (c), and from the ensemble of three segmentation branches for the architecture of Figure 3.6 (d). Our experiments show that these architectures of the Cascade Mask R-CNN outperform the Mask R-CNN.

3.6 Experimental Results

In this section, we present an extensive evaluation of the Cascade R-CNN detector.

3.6.1 Experimental Set-up

Experiments were performed over multiple datasets and baseline network architectures.

Datasets: The bulk of the experiments was performed on MS-COCO 2017 [98], which contains ∼118k images for training, 5k for validation (`val`) and ∼20k for testing without provided annotations (`test-dev`). The COCO average precision (AP) measure averages AP across IoU thresholds from 0.5 to 0.95, with an interval of 0.05. It measures detection performance at various qualities, encouraging high quality detection results, as discussed in Section 3.3.3. All models were trained on the COCO training set and evaluated on the `val` set. Final results are also reported on the `test-dev` set for fair comparison with the state-of-the-art. To assess the robustness and generalization ability of the Cascade R-CNN, experiments were also performed on Pascal VOC [36], KITTI [43], CityPersons [193] and WiderFace [182]. Instance segmentation was also evaluated on COCO, using the same evaluation metrics as object detection. The only difference is that the IoU is computed with respect to the mask rather than a bounding box.

Implementation Details: All regressors are class agnostic for simplicity. All Cascade R-CNN detection stages have the same architecture, which is the detection head of the baseline detector. Unless otherwise noted, the Cascade R-CNN is implemented with four stages: one RPN and three detection heads with thresholds $U = \{0.5, 0.6, 0.7\}$. The sampling of the first detection stage follows [50, 140]. In subsequent stages, resampling is implemented by using *all* the regressed outputs from the previous stage, as discussed in Section 3.4.3. No data augmentation was used except standard horizontal image flipping. Inference was performed at a single image scale, with no further bells and whistles. All baseline detectors were reimplemented with Caffe [76], using the same codebase, for fair comparison. Some experiments with the FPN and Mask R-CNN baselines were implemented on the Detectron platform.

Baseline Networks: To test the versatility of the Cascade R-CNN, experiments were performed with multiple popular baselines: Faster R-CNN and MS-CNN of Chapter 2 with VGG-Net [150] backbone, R-FCN [24] and FPN [99] with ResNet backbones [61], for the task of object detection, and Mask R-CNN [58] with ResNet backbones for instance segmentation. These baselines have a wide range of performances. Unless noted, their default settings were used. End-to-end training was used instead of multi-step training.

In Faster R-CNN, the network head has two fully connected layers. To reduce parameters, [55] was used to prune less important connections. 2048 units were retained per fully connected layer and dropout layers were removed. These changes have negligible effect on detection performance. Training started with a learning rate of 0.002, which was reduced by a factor of 10 at 60k and 90k iterations, and stopped at 100k iterations, on 2 synchronized GPUs, each holding 4 images per iteration. 128 RoIs were used per image.

The R-FCN adds a convolutional, a bounding box regression, and a classification layer to the ResNet. For this baseline, all Cascade R-CNN heads have this structure. Online hard negative mining [149] was not used. Training started with a learning rate of 0.003, which was decreased by a factor of 10 at 160k and 240k iterations, and stopped at 280k iterations, on 4 synchronized

GPUs, each holding one image per iteration. 256 RoIs were used per image.

Since official source code was not publicly available for the FPN when we performed our original experiments [12], the implementation details were somewhat different from those later made available in the Detectron implementation. RoIAlign [58] was used for a stronger baseline. This is denoted as FPN+ and was used in all ablation studies, with the ResNet-50 as a backbone as usual. Training used a learning rate of 0.005 for 120k iterations and 0.0005 for the next 60k iterations, on 8 synchronized GPUs, each holding one image per iteration. 256 RoIs were used per image. We have also reimplemented the Cascade R-CNN of FPN on Detectron platform, when it is publicly available.

The MS-CNN, introduced in Chapter 2, is a good multi-scale object detector for specific object categories, e.g. vehicle, pedestrian, face, etc. It was used as baseline detector for experiments on KITTI, CityPersons and WiderFace. For this baseline, the Cascade R-CNN adopted the same two-step training strategy of the MS-CNN: proposal sub-network trained first and then joint end-to-end training. All detection heads were only added at the second step, where the learning rate was initially 0.0005, decreased by a factor of 10 at 10k and 20k iterations and stopped at 25k iterations, on one GPU of batch size 4 images.

The Mask R-CNN was used as baseline for instance segmentation. The default Detectron implementation was adopted, using the `1x` learning schedule. Training started with a learning rate of 0.02, which was reduced by a factor of 10 at 60k and 80k iterations, and stopped at 90k iterations, on 8 synchronized GPUs, each holding 2 images per iteration. 512 RoIs were used per image.

3.6.2 Quality Mismatch

An initial set of experiments was designed to evaluate the impact of the mistmatch between proposal and detector quality on detection performance. Figure 3.7 (a) shows the AP curves of three individually trained detectors of increasing IoU threshold in $U = \{0.5, 0.6, 0.7\}$. The

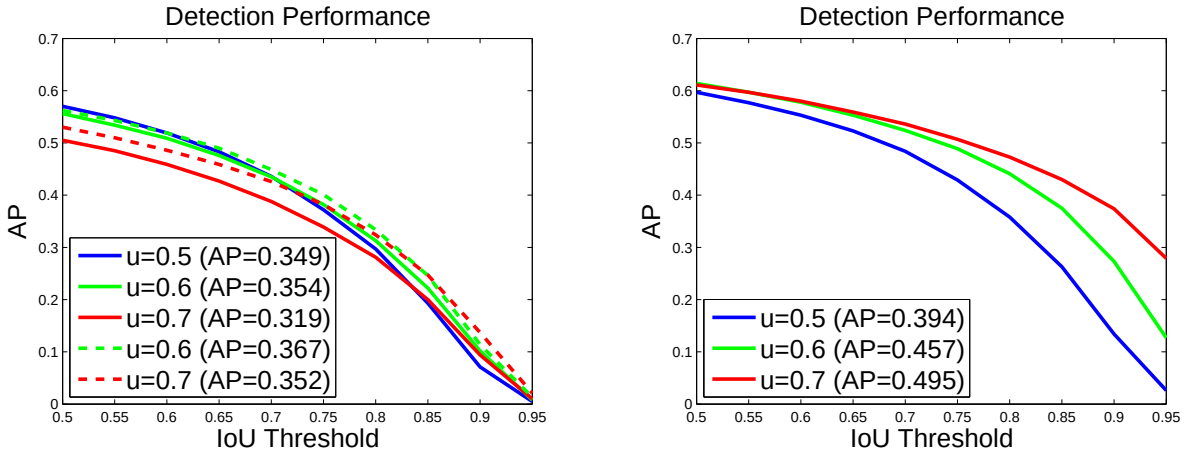


Figure 3.7: (a) detection performance of individually trained detectors, with their own proposals (solid curves) or Cascade R-CNN stage proposals (dashed curves). (b) results of adding ground truth to the proposal set.

detector of $u = 0.5$ outperforms the detector of $u = 0.6$ at low IoU levels, but underperforms it at higher levels. However, the detector of $u = 0.7$ underperforms the other two. To understand why this happens, we changed the quality of the proposals at inference. Figure 3.7 (b) shows the results obtained when ground truth bounding boxes were added to the set of proposals. While all detectors improved, the detector of $u = 0.7$ had the largest gains, and the best performance for almost all IoU levels. These results suggest two conclusions. First, the commonly used $u = 0.5$ threshold is not effective for precise detection, simply more robust to low quality proposals. Second, precise detection requires hypotheses that match the detector quality.

Next, the original proposals were replaced by the Cascade R-CNN proposals of higher quality ($u = 0.6$ and $u = 0.7$ used the 2nd and 3rd stage proposals, respectively). Figure 3.7 (a) suggests that the performance of the two detectors is significantly improved when the quality of the test proposals matches the detector quality. Testing Cascade R-CNN detectors of different qualities at all cascade stages produced similar observations. Figure 3.8 shows that each detector was improved by the use of more precise hypotheses, with higher quality detectors exhibiting larger gains. For example, the detector of $u = 0.7$ performed poorly for the low quality proposals

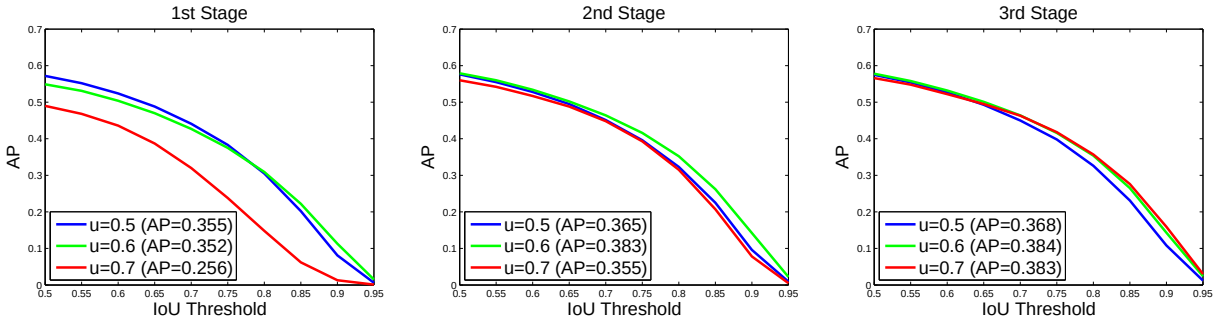


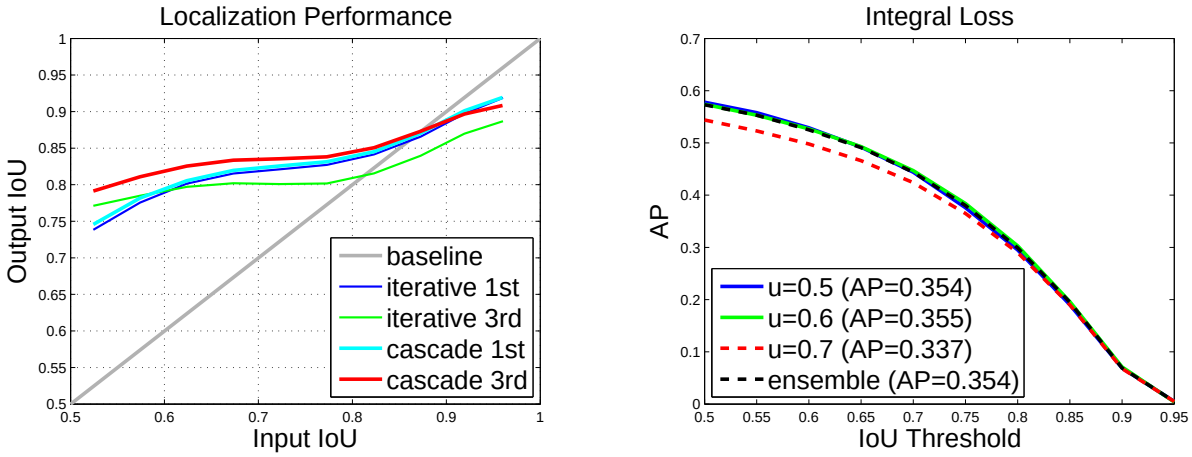
Figure 3.8: Detection performance of all Cascade R-CNN detectors at all cascade stages.

of the 1st stage, but much better for the more precise hypotheses available at the deeper cascade stages. The jointly trained detectors of Figure 3.8 also outperformed the individually trained detectors of Figure 3.7 (a), even when the same proposals were used. This indicates that the detectors are better trained within the Cascade R-CNN architecture.

3.6.3 Comparison with *Iterative BBox* and *Integral Loss*

In this section, we compare the Cascade R-CNN to the *iterative BBox* and *integral loss* detectors. *Iterative BBox* was implemented by applying the detection head of FPN+ baseline iteratively at inference, three times. The *integral loss* detector was implemented with three classification heads, using $U = \{0.5, 0.6, 0.7\}$.

Localization: The localization performances of cascade regression and *iterative BBox* are compared in Figure 3.9 (a). The use of a single regressor degrades localization for hypotheses of high IoU. This effect accumulates when the regressor is applied iteratively, as in *iterative BBox*, and performance actually drops with iteration number. Note the very poor performance of *iterative BBox* after 3 iterations. On the contrary, the cascade regressor has better performance at later stages, outperforming *iterative BBox* at almost all IoU levels. Note that, although cascade regression can slightly degrade high input IoUs, e.g. IoU$>$0.9, this decrease is negligible because, as shown in Figure 3.4, the number of hypotheses with such high IoUs is extremely small.



(a) (b)

Figure 3.9: (a) localization performance of *iterative BBox* and Cascade R-CNN regressors. (b) detection performance of the individual classifiers of the *integral loss* detector.

Table 3.1: Comparison of the Cascade R-CNN with *iterative BBox* and *integral loss* detectors.

	AP	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
FPN+ baseline	34.9	57.0	51.9	43.6	29.7	7.1
Iterative BBox	35.4	57.2	52.1	44.2	30.4	8.1
Integral Loss	35.4	57.3	52.5	44.4	29.9	6.9
Cascade R-CNN	**38.9**	**57.8**	**53.4**	**46.9**	**35.8**	**15.8**

Integral Loss: Figure 3.9 (b) summarizes the detection performances of all classifiers of the *integral loss* detector, sharing a single regressor. The classifier of $u = 0.6$ is the best at all IoU levels, with $u = 0.7$ producing the worst results. The ensemble of all classifiers shows no visible gain.

Table 3.1 shows that both *iterative BBox* and *integral loss* marginally improve on the baseline detector, and are not effective for high quality detection. On the other hand, the Cascade R-CNN achieves the best performance at all IoU levels. As expected, the gains are mild for low IoUs, e.g. 0.8 for AP_{50}, but significant for the higher ones, e.g. 6.1 for AP_{80} and 8.7 for AP_{90}. Note that high quality object detection was rarely explored before this work. These experiments show that 1) it has more room for improvement than low quality detection, which focuses on AP_{50}, and 2) the overall AP can be significantly improved if it is effectively addressed.

Table 3.2: Stagewise performance of the Cascade R-CNN. $\overline{1 \sim 3}$ indicates an ensemble result, obtained by averaging the three classifier probabilities for 3rd stage proposals.

test stage	AP	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
1	35.5	57.2	52.4	44.1	30.5	8.1
2	38.3	57.9	53.4	46.4	35.2	14.2
3	38.3	56.6	52.2	46.3	35.7	**15.9**
$\overline{1 \sim 2}$	38.5	**58.2**	**53.8**	46.7	35.0	14.0
$\overline{1 \sim 3}$	**38.9**	57.8	53.4	**46.9**	**35.8**	15.8
FPN+ baseline	34.9	57.0	51.9	43.6	29.7	7.1

3.6.4 Ablation Experiments

A few ablation experiments were run to enable a better understanding of the Cascade R-CNN.

Stage-wise Comparison: Table 3.2 summarizes stagewise performance. Note that the first stage already outperforms the baseline detector, due to the benefits of multi-stage multi-task learning. Since deeper cascade stages prefer higher quality localization, they encourage the learning of features conducive to it. This benefits the earlier cascade stages, due to the feature sharing by the backbone network. The second stage improves performance substantially, and the third is equivalent to the second. This differs from the *integral loss* detector, where the higher IoU classifier is relatively weak. While the former (later) stage is better at low (high) IoU metrics, the ensemble of all classifiers is the best overall.

IoU Thresholds: A Cascade R-CNN was trained using IoU threshold $u = 0.5$ for all heads. In this case, the stages differ only in the hypotheses at their input. Each stage is trained with the corresponding hypotheses, i.e. accounting for the distribution changes of Figure 3.5. The first row of Table 3.3 shows that this cascade improves on the baseline detector. This supports the claim that stages should be optimized for the corresponding sample distributions. The second row shows performance that improves further when the threshold u increases across stages. As discussed in Section 3.4.3, the detector becomes more selective against close false positives and *specialized* to the more precise hypotheses.

Table 3.3: Ablation experiments. "IoU↑" indicates increasing IoU thresholds, "*update statistics*" updating regression statistics, and "*stage loss*" weighting of stage losses.

IoU↑	*update statistics*	*stage loss*	AP	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
		decay	36.8	57.8	52.9	45.4	32.0	10.7
✓		*decay*	38.5	58.4	54.1	47.1	35.0	13.1
	✓	*decay*	37.5	57.8	53.1	45.5	33.3	13.1
✓	✓	*decay*	38.9	57.8	53.4	46.9	35.8	15.8
✓	✓	*avg*	38.9	57.5	53.4	46.9	35.8	16.2

Regression Statistics: In Section 3.3.1, we saw that the distance vector Δ is normalized by the regression statistics (mean and variance), as in (3.5). In the Cascade R-CNN, these statistics are updated stage by stage, as illustrated in Figure 3.5. Updating the statistics of (3.5) in deeper stages helps the effective multi-task learning of classification and regression. Empirically, the learning is not very sensitive to the exact values of these statistics. For simplicity, we set $\mu = 0$ for all stages, $\Sigma = (\sigma_x, \sigma_y, \sigma_w, \sigma_h) = (0.1, 0.1, 0.2, 0.2)$ for the first stage, $\Sigma/2$ for the second, and $\Sigma/3$ for the third, in all of our experiments. The third and fourth row of Table 3.3 show that this is beneficial, when compared to using the statistics of the first stage in all stages (the first and second row).

Stage Losses: The Cascade R-CNN has multiple detection heads, each with its own loss. We have explored two schemes to combine these losses: *decay* and *avg*. In *avg*, the loss of stage t receives a weight $w_t = 1/T$, where T is the number of stages. In *decay*, the weight is $w_t = 1/2^{t-1}$. For both schemes, the learning rate of the head parameters of stage t is rescaled by $1/w_t$, to ensure that these are sufficiently trained. No rescaling is needed for the backbone network parameters, since they receive gradients from all stages. Table 3.3 shows that 1) *avg* has somewhat better performance for high quality metrics, but worse for low quality ones, and 2) the two methods have similar overall AP. The *decay* scheme is used in the remainder of the chapter.

Number of Stages: Table 3.4 summarizes the impact of the number of stages in the Cascade R-CNN performance. Adding a second stage significantly improves the baseline detector.

Table 3.4: The impact of the number of stages in Cascade R-CNN.

# stages	test stage	AP	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
1	1	34.9	57.0	51.9	43.6	29.7	7.1
2	$\overline{1\sim2}$	38.2	**58.0**	**53.6**	46.7	34.6	13.6
3	$\overline{1\sim3}$	**38.9**	57.8	53.4	**46.9**	35.8	15.8
4	$\overline{1\sim3}$	**38.9**	57.4	53.2	46.8	**36.0**	16.0
4	$\overline{1\sim4}$	38.6	57.2	52.8	46.2	35.5	**16.3**

Three detection stages still produce non-trivial improvement, but the addition of a 4^{th} stage ($u = 0.75$) has a slight performance decrease. Note, however, that while the overall AP degrades, the four-stage cascade has the best performance at high IoU levels. The three-stage cascade achieves the best trade-off between cost and AP performance, and is used in the remaining experiments.

3.6.5 Comparison with the state-of-the-art

An implementation of the Cascade R-CNN, based on the FPN+ detector and the ResNet-101 backbone, is compared to state-of-the-art *single-model* detectors in Table 3.5[6]. The settings are those of Section 3.6.1, but training used 280k iterations, with learning rate decreased at 160k and 240k iterations. The number of RoIs was also increased to 512. The top of the table reports to one-stage detectors, the middle to two-stage, and the bottom to multi-stage (3-stages+RPN for the Cascade R-CNN). Note that *all* the compared state-of-the-art detectors are trained with $u = 0.5$.

An initial observation is that our FPN+ implementation is better than the original FPN [99], providing a very strong baseline. Nevertheless, the extension from FPN+ to Cascade R-CNN improved performance by $\sim$4 points. In fact, the vanilla Cascade R-CNN, without any bells and whistles, outperformed almost all *single-model* detectors under all evaluation metrics. This includes the COCO challenge 2016 winner G-RMI [68], the recent Deformable R-FCN [25], RetinaNet [100], Mask R-CNN [58], RelationNet[65], DetNet[93], CornerNet [87], etc. Note

[6]Some detectors are omitted in this comparison because their *single-model* results on COCO `test-dev` are not publicly available.

Table 3.5: Performance of state-of-the-art *single-model* detectors on COCO `test-dev`. Entries denoted by * and * use enhancements at training and inference, respectively.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv2 [137]	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [104]*	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8
RetinaNet [100]*	ResNet-101	39.1	59.1	42.3	21.8	42.7	50.2
CornerNet [87]**	Hourglass-104	42.1	57.8	45.3	20.8	44.8	56.7
Faster R-CNN+++ [61]**	ResNet-101	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [99]	ResNet-101	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN w FPN+ (ours)	ResNet-101	38.8	61.1	41.9	21.3	41.8	49.8
G-RMI [68]**	Inception-ResNet-v2	41.6	62.3	45.6	24.0	43.9	55.2
Deformable R-FCN [25]**	Aligned-Inception-ResNet	37.5	58.0	40.8	19.4	40.1	52.5
Mask R-CNN [58]	ResNet-101	38.2	60.3	41.7	20.1	41.1	50.2
RelationNet [65]	ResNet-101	39.0	58.6	42.9	-	-	-
DetNet [93]	DetNet-59	40.3	62.1	43.8	23.6	42.6	50.0
SNIP [151]**	DPN-98	45.7	67.3	51.1	29.3	48.8	57.1
AttractioNet [47]*	VGG16+Wide ResNet	35.7	53.4	39.3	15.6	38.0	52.7
Cascade R-CNN	ResNet-101	42.8	62.1	46.3	23.7	45.5	55.2
Cascade R-CNN**	ResNeXt-152	50.9	69.0	55.8	33.4	53.5	63.3

Table 3.6: Performance of Cascade R-CNN implementations with multiple detectors. All speeds are reported per image on a single Titan Xp GPU.

	backbone	cascade	train speed	test speed	model size	`val` (5k)						`test-dev` (20k)					
						AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN	VGG	✗	0.12s	0.075s	278M	23.6	43.9	23.0	8.0	26.2	35.5	23.5	43.9	22.6	8.1	25.1	34.7
		✓	0.14s	0.115s	704M	27.0	44.2	27.7	8.6	29.1	42.2	26.9	44.3	27.8	8.3	28.2	41.1
R-FCN	ResNet-50	✗	0.19s	0.07s	133M	27.0	48.7	26.9	9.8	30.9	40.3	27.1	49.0	26.9	10.4	29.7	39.2
		✓	0.24s	0.075s	184M	31.1	49.8	32.8	10.4	34.4	48.5	30.9	49.9	32.6	10.5	33.1	46.9
R-FCN	ResNet-101	✗	0.23s	0.075s	206M	30.3	52.2	30.8	12.0	34.7	44.3	30.5	52.9	31.2	12.0	33.9	43.8
		✓	0.29s	0.083s	256M	33.3	52.0	35.2	11.8	37.2	51.1	33.3	52.6	35.2	12.1	36.2	49.3
FPN+	ResNet-50	✗	0.30s	0.095s	165M	36.5	58.6	39.2	20.8	40.0	47.8	36.5	59.0	39.2	20.3	38.8	46.4
		✓	0.33s	0.115s	272M	40.3	59.4	43.7	22.9	43.7	54.1	40.6	59.9	44.0	22.6	42.7	52.1
FPN+	ResNet-101	✗	0.38s	0.115s	238M	38.5	60.6	41.7	22.1	41.9	51.1	38.8	61.1	41.9	21.3	41.8	49.8
		✓	0.41s	0.14s	345M	42.7	61.6	46.6	23.8	46.2	57.4	42.8	62.1	46.3	23.7	45.5	55.2

some of these methods leverage several training or inference enhancements, e.g. multi-scale, soft NMS [7], etc, making the comparison very unfair. Finally, compared to the previously best multi-stage detector on COCO, AttractioNet [47], also using many enhancements, the vanilla Cascade R-CNN has a gain of 7.1 points.

The only detector that outperforms the Cascade R-CNN in Table 3.5 is SNIP [151], which uses multi-scale training and inference, a larger input size, a stronger backbone, Soft NMS, and some other enhancements. For a more fair comparison, we implemented the Cascade R-CNN with multi-scale training/inference, a stronger backbone (ResNeXt-152 [172]), mask supervision, etc. This enhanced Cascade R-CNN surpassed SNIP by 5.2 points. It also outperforms the *single-model* MegDet detector (50.6 mAP), which won the COCO challenge in 2017 and uses many other enhancements [131]. The Cascade R-CNN is conceptually straightforward, simple to implement, and can be combined, in a plug and play manner, with many detector architectures.

3.6.6 Generalization Capacity

To more thoroughly test this claim, a three-stage Cascade R-CNN was implemented with three baseline detectors: Faster R-CNN, R-FCN, and FPN+. All settings are as discussed above, with the variations discussed in Section 3.6.5 for the FPN+ detector. Table 3.6 presents a comparison of the AP performance of the three detectors.

Detection Performance: Again, our implementations are better than the original detectors [140, 24, 99]. Still, the Cascade R-CNN improves on all baselines by 2$\sim$4 points, independently of their strength. Similar gains are observed for `val` and `test-dev`. These results show that the Cascade R-CNN is widely applicable across detector architectures.

Parameters and Timing: The number of Cascade R-CNN parameters increases with the number of stages. The increase is linear and proportional to the parameter cardinality of the baseline detector head. However, because the head has much less computation than the backbone network, the Cascade R-CNN has small computational overhead, at both training and testing.

Table 3.7: Performance of various implementations of the Cascade R-CNN with the FPN detector on Detectron, using the `1x` schedule.

backbone	cascade	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Fast ResNet-50	✗	36.4	58.4	39.3	20.3	39.8	48.1
	✓	40.5	58.7	43.9	21.5	43.6	54.9
ResNet-50	✗	36.7	58.4	39.6	21.1	39.8	48.1
	✓	40.9	59.0	44.6	22.5	43.6	55.3
ResNet-101	✗	39.4	61.2	43.4	22.6	42.9	51.4
	✓	42.8	61.4	46.8	24.1	45.8	57.4
ResNeXt-101	✗	41.3	63.7	44.7	25.5	45.3	52.9
	✓	44.7	63.7	48.8	26.3	48.4	58.6
ResNet-50-GN	✗	38.4	59.9	41.7	22.2	41.2	50.0
	✓	42.2	60.6	45.8	24.7	45.2	55.7
ResNet-101-GN	✗	39.9	61.3	43.3	23.6	42.8	52.3
	✓	43.8	62.2	47.6	26.2	47.2	57.7

This is shown in Table 3.6.

Codebase and Backbone: The Cascade R-CNN of FPN was also reimplemented on the Detectron codebase [49] with various backbone networks. Table 3.7 summarizes these experiments, showing very consistent improvements (3$\sim$4 points) across backbones. The Cascade R-CNN has also been independently reproduced by other research groups, on PyTorch and TensorFlow. These again show that the Cascade R-CNN can provide reliable gains across detector architectures, backbones, codebases, and implementations.

Fast R-CNN: As shown in Figure 3.3 (b), the Cascade R-CNN is not limited to the standard Faster R-CNN architecture. To test this, we trained the Cascade R-CNN in the way of the Fast R-CNN, using pre-collected proposals. The results of Table 3.7 show that the gains of the Cascade R-CNN hold for frameworks other than the Faster R-CNN.

Group Normalization: Group normalization (GN) [167] is a recent normalization technique, published after the Cascade R-CNN. It addresses the problem that batch normalization (BN) [73] must be frozen for object detector training, due to the inaccurate statistics that can be derived from small batch sizes [131]. GN, an alternative to BN that is independent of batch size, has comparable performance to large-batch synchronized BN. Table 3.7 shows that the Cascade

Table 3.8: Proposal recall of Cascade R-CNN stages.

stage	AP^{100}	AP^{100}_s	AP^{100}_m	AP^{100}_l	AP^{1k}	AP^{1k}_s	AP^{1k}_m	AP^{1k}_l
FPN	47.8	32.2	54.9	65.2	59.1	48.0	66.3	68.4
1	46.8	31.0	53.8	64.8	58.7	47.6	65.9	68.2
2	55.3	35.1	61.1	82.5	70.7	55.2	77.7	88.1
3	56.5	36.1	62.4	84.1	71.4	55.5	78.1	89.8

R-CNN with GN has similar gains to those obersved for the other architectures. This suggests that the Cascade R-CNN will continue to be useful even as architectural enhancements continue to emerge in the literature.

3.6.7 Proposal Evaluation

Table 3.8 summarizes the proposal recall performance of a Cascade R-CNN implemented with the FPN detector and ResNet-50 backbone. The first Cascade R-CNN stage has proposal recall close to that of the FPN baseline. The addition of a bounding box regression stage improves recall significantly, e.g. from 59.1 to 70.7 for AP^{1k} and close to 20 points for AP^{1k}_l. This shows that the additional bounding box regression is very effective at improving proposal recall performance. The addition of a third stage has a smaller but non-negligible gain. Note that the COCO recall is the mean recall over IoU thresholds [0.5:0.05:0.95]. This high proposal recall performance secures the later high-quality object detection task.

3.6.8 Instance Segmentation by Cascade Mask R-CNN

Table 3.9 summarizes the instance segmentation performance of the Cascade Mask R-CNN strategies of Figure 3.6. These experiments, use the Mask R-CNN, implemented on Detectron with `1x` schedule as baseline. All three strategies improve on baseline performance, although with smaller gains than object detection (see Table 3.2), especially at high quality. For example, the AP_{90} improvement of 8.7 points for object detection falls to 1.8 points, showing that plenty of room is left for improving high quality instance segmentation. Comparing strategies, (c)

Table 3.9: The instance segmentation comparison among three strategies of the Cascade Mask R-CNN.

	AP	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
Mask R-CNN baseline	33.9	55.5	49.8	41.4	28.6	8.3
strategy of Figure 3.6 (b)	35.0	56.3	50.8	43.0	30.0	9.3
strategy of Figure 3.6 (c)	35.4	56.4	50.9	43.2	31.0	10.1
strategy of Figure 3.6 (d)	35.5	56.5	51.2	43.4	30.8	10.0

outperforms (b). This is because (b) trains the mask head in the first stage but tests after the last stage, leading to a mask prediction mismatch. This mismatch is reduced by (c). The addition of a mask branch to each stage by strategy (d) does not have noticeable benefits over (c), but requires much more computation and memory. Strategy (b) has the best trade-off between cost and AP performance, and is used in the remainder of the chapter.

To evaluate the instance segmentation robustness of the Cascade Mask R-CNN, several backbone networks are compared in Table 3.10. Since this architecture can detect objects, detection results are also shown. Note that the additional mask supervision makes these better than those of Table 3.7. The gains of the Cascade Mask R-CNN are very consistent for all backbone networks. Even when the strongest model, ResNeXt-152 [172], is used with training data augmentation and `1.44x` schedule, the Cascade Mask R-CNN has a gain of 2.9 points for detection and 1.0 point for instance segmentation. Adding inference enhancements, the gains are still 2.1 points for detection and 0.8 points for instance segmentation. This robustness explains why the Cascade R-CNN was widely used in the COCO challenge 2018, where the task is instance segmentation, not object detection.

3.6.9 Results on PASCAL VOC

The Cascade R-CNN was further tested on the PASCAL VOC dataset [36]. Following [140, 104], the models were trained on VOC2007 and VOC2012 `trainval` (16,551 images) and tested on VOC2007 `test` (4,952 images). Two detector architectures were evaluated: Faster

Table 3.10: Performance of the Cascade Mask R-CNN on multiple backbone networks on COCO 2017 `val`. * and ⋆ denotes enhancement techniques at training and inference, respectively, as in [49].

backbone	cascade	Object Detection						Instance Segmentation					
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet-50	✗	37.7	59.2	40.9	21.4	40.8	49.8	33.9	55.8	35.8	14.9	36.3	50.9
	✓	41.3	59.4	45.3	23.2	43.8	55.8	35.4	56.4	37.7	15.9	37.7	53.6
ResNet-101	✗	40.0	61.8	43.7	22.5	43.4	52.7	35.9	58.3	38.0	15.9	38.9	53.2
	✓	43.3	61.7	47.2	24.2	46.3	58.2	37.1	58.6	39.8	16.7	39.7	55.7
ResNet-50-GN	✗	39.2	60.5	42.9	22.9	42.2	50.6	34.9	57.1	36.9	16.0	37.7	51.2
	✓	42.9	60.7	46.6	25.1	45.9	56.7	36.6	57.7	39.2	16.8	39.3	54.5
ResNet-101-GN	✗	41.1	62.1	45.1	23.6	44.3	53.1	36.3	58.9	38.5	16.2	39.4	53.6
	✓	44.8	62.8	48.8	26.4	48.0	58.7	38.0	59.8	40.8	18.1	40.7	56.0
ResNeXt-101	✗	42.1	64.1	45.9	25.6	45.9	54.4	37.3	60.3	39.5	17.8	40.3	55.5
	✓	45.8	64.1	50.3	27.2	49.5	60.1	38.6	60.6	41.5	18.5	41.3	57.2
ResNeXt-152*	✗	45.2	66.9	49.7	28.5	49.4	56.8	39.7	63.5	42.4	19.8	42.9	57.3
	✓	48.1	66.7	52.6	29.3	52.2	62.1	40.7	63.7	43.8	19.9	44.0	59.1
ResNeXt-152**	✗	48.1	68.3	52.9	32.6	51.8	61.3	41.5	65.1	44.7	22.0	44.8	59.8
	✓	50.2	68.2	55.0	33.1	53.9	64.2	42.3	65.4	45.8	21.9	45.7	60.9

Table 3.11: Detection results on PASCAL VOC 2007 `test`.

	backbone	cascade	AP	AP_{50}	AP_{75}
Faster R-CNN	AlexNet	✗	29.4	63.2	23.7
		✓	38.9	66.5	40.5
Faster R-CNN	VGG	✗	42.9	76.4	44.1
		✓	51.2	79.1	56.3
R-FCN	RetNet-50	✗	44.8	77.5	46.8
		✓	51.8	78.5	57.1
R-FCN	ResNet-101	✗	49.4	79.8	53.2
		✓	54.2	79.6	59.2

R-CNN (with AlexNet and VGG-Net backbones) and R-FCN (with ResNet-50 and ResNet-101). Training details were as discussed in Section 3.6.1, and both AlexNet and VGG-Net were pruned. More specifically, Faster R-CNN (R-FCN) training started with a learning rate of 0.001 (0.002), which was reduced by a factor of 10 at 30k (60k) and stopped at 45k (90k) iterations. Since the standard VOC evaluation metric (AP at IoU of 0.5) is fairly saturated, and the focus of this work is high quality detection, the COCO metrics were used for evaluation[7]. Table 3.11 summarizes the performance of all detectors, showing that the Cascade R-CNN significantly improves the overall AP in all cases. These results are further evidence for the robustness of the Cascade R-CNN.

3.6.10 Additional Results on other Datasets

Beyond generic object detection datasets, the Cascade R-CNN was tested on some specific object detection tasks, including KITTI [43], CityPerson [193] and WiderFace [182]. The MS-CNN, a detector of strong performance on these tasks, was used as baseline for all of them.

KITTI: One of the most popular datasets for autonomous driving, KITTI contains 7,481 training/validation images, and 7,518 for testing with held annotations. The 2D object detection task contains three categories: car, pedestrian, and cyclist. Evaluation is based on the VOC AP at IoU of 0.7, 0.5, and 0.5 for the three categories, respectively. Since the focus of this work

[7]The PASCAL VOC annotations were transformed to COCO format, and the COCO toolbox used for evaluation. Results are different from the standard VOC evaluation.

Table 3.12: MS-CNN detection results for the car class on KITTI test set.

	cascade	Easy	Moderate	Hard
AP_{70}	✗	90.22	89.08	76.50
	✓	90.68	89.95	78.40

Table 3.13: MS-CNN detection results on CityPersons validation set.

	cascade	Reasonable	Small	Heavy	All
MR_{50}	✗	13.07	40.42	53.55	38.74
	✓	11.96	38.37	49.41	36.83
MR_{75}	✗	38.23	59.84	85.06	65.56
	✓	28.45	56.24	81.86	58.24

is high quality detection, the Cascade R-CNN was only tested on the car category. As shown in Table 3.12, it improved the baseline by 0.87 points for the `Moderate`, and 1.9 points for the `Hard` regime, on the test set. These improvements are nontrivial, given that MS-CNN is a strong detector and the KITTI `car` detection task is fairly saturated.

CityPersons: CityPersons is a recently published pedestrian detection dataset, collected across multiple European cities. It contains 2,975 training and 500 validation images, and 1,575 images for testing with held annotations. Evaluation is based on miss-rate (MR) at IoU=0.5. We also report results for MR at IoU=0.75, which is more commensurate with high quality detection. This is consistent with a recent trend to adopt the stricter COCO metric for pedestrian and face detection, see e.g. the Wider Challenge 2018. Table 3.13 compares the validation set performance of the Cascade R-CNN with that of the baseline MS-CNN (performances on validation and test sets are usually equivalent on this dataset). The Cascade R-CNN has large performance gains, especially for the stricter evaluation metric. For example, it improves the baseline performance by $\sim$10 points on the `Reasonable` set at MR_{75}.

WiderFace: One of the most challenging face detection datasets, mainly due to its diversity in scale, pose and occlusion, WiderFace contains 32,203 images with 393,703 annotated faces, of which 12,880 are used for training, 3,226 for validation, and the remainder for testing with held annotations. Evaluation is based on the VOC AP at IoU=0.5 on three subsets, `easy`,

Table 3.14: MS-CNN Detection results on WiderFace validation set.

	cascade	Easy	Medium	Hard
AP_{50}	✗	91.1	90.6	81.0
	✓	91.3	90.3	81.1
AP_{75}	✗	59.7	61.3	40.7
	✓	68.7	66.3	42.8

`medium` and `hard`, of different detection difficulty. Again, we have used AP at IoU=0.5 and IoU=0.75 and evaluation on the validation set. Table 3.14 shows that, while the Cascade R-CNN is close to the baseline MS-CNN for AP_{50}, it significantly boosts its performance for AP_{75}. The gain is smaller on the `hard` than on the `easy` and `medium`, because the former contains mainly very small and heavily occluded faces, for which high quality detection is difficult. This observation mirrors the COCO experiments of Table 3.6, where improvements in AP_S are smaller than for AP_L.

3.7 Conclusion

In this chapter, we have proposed a multi-stage object detection framework, the Cascade R-CNN, for high quality object detection, a rarely explored problem in the detection literature. This architecture was shown to overcome the high quality detection challenges of overfitting during training and quality mismatch during inference. This is achieved by training stages sequentially, using the output of one to train the next, and the same cascade is applied at inference. The Cascade R-CNN was shown to achieve very consistent performance gains on multiple challenging datasets, including COCO, PASCAL VOC, KITTI, CityPersons, and WiderFace, for both generic and specific object detection. These gains were also observed for many object detectors, backbone networks, and techniques for detection and instance segmentation. We thus believe that the Cascade R-CNN can be useful for many future object detection and instance segmentation research efforts.

3.8 Acknowledgements

Chapter 3 is, in full, based on the material as it appears in the publication of “Cascade R-CNN: High Quality Object Detection and Instance Segmentation”, Zhaowei Cai, and Nuno Vasconcelos, to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2020. The dissertation author was the primary investigator and author of this material.

Chapter 4

Domain-Universal Object Detection

4.1 Introduction

There has been significant progress in object detection in recent years [50, 140, 10, 99, 58, 12], powered by the availability of challenging and diverse object detection datasets, e.g. PASCAL VOC [36], COCO [98], KITTI [43], WiderFace [182], etc. However, existing detectors are usually *domain-specific*, e.g. trained and tested on a single dataset. This is partly due to the fact that object detection datasets are diverse and there is a nontrivial domain shift between them. As shown in Figure 4.1, detection tasks can vary in terms of categories (human face, horse, medical lesion, etc.), camera viewpoints (images taken from aircrafts, autonomous vehicles, etc.), image styles (comic, clipart, watercolor, medical), etc. In general, high detection performance requires a detector specialized on the target dataset.

This poses a significant problem for practical applications, which are not usually restricted to any one of the domains of Figure 4.1. Hence, there is a need for systems capable of detecting objects regardless of the domain in which images are collected. A simple solution is to design a *specialized* detector for each domain of interest, e.g. use D detectors trained on D datasets, and load the detector specialized to the domain of interest at each point in time. This, however, may be impractical, for two reasons. First, in most applications involving autonomous systems the domain of interest can change frequently and is not necessarily known a priori. Second, the overall model size increases linearly with the number of domains D. A recent trend, known as general AI, is to request that a single universal model solves multiple tasks [80, 84, 188], or the same task over multiple domains [135, 6]. However, existing efforts in this area mostly address image classification, rarely targeting the problem of object detection. The fact that modern object detectors are complex systems, composed of a backbone network, proposal generator, bounding box regressor, classifier, etc., makes the design of a universal object detector much more challenging than a universal image classifier.

In this work, we consider the design of an object detector capable of operating over

Figure 4.1: Samples of our universal object detection benchmark.

multiple domains. We begin by establishing a new universal object detection benchmark, denoted as UODB, consisting of 11 diverse object detection datasets (see Figure 4.1). This is significantly more challenging than the Decathlon [135] benchmark for multi-domain recognition. To the best of our knowledge, we are the first to attack universal object detection using deep learning. We expect this new benchmark will encourage more efforts in the area. We then propose a number of

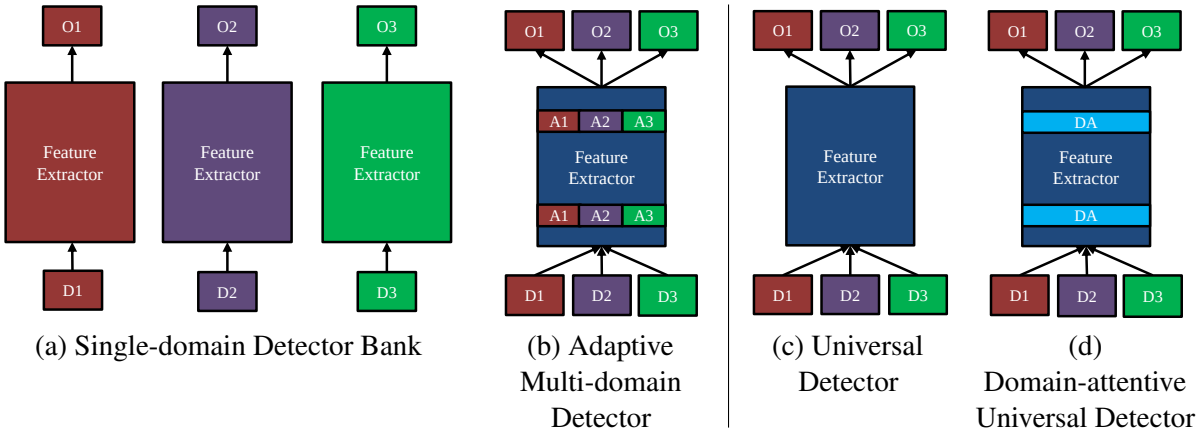


Figure 4.2: Multi-domain and universal object detectors for three domains. "D" is the domain, "O" the output, "A" domain-specific adapter, and "DA" the proposed domain attention module. The blue color and the DA are domain-universal, but the other colors domain-specific.

architectures, shown in Figure 4.2, to address the universal/multi-domain detection problem.

The two architecture on the left of Figure 4.2 are multi-domain detectors, which require prior knowledge of the domain of interest. The two architectures on the right are universal detectors, with no need for such knowledge. When operating on an unknown domain, the multi-domain detector have to repeat the inference process with different sets of domain-specific parameters, while the universal detector performs inference only once. The detector of Figure 4.2 (a) is a bank of domain-specific detectors, with no sharing of parameters/computations. Multi-domain learning (MDL) [79, 120, 83, 183, 78, 34] improves on this, by sharing parameters across various domains, and adding small domain-specific layers. In [135, 6], expensive convolutional layers are shared and complemented with light-weight domain-specific *adaptation layers*. Inspired by these, we propose a new class of light adapters for detection, based on the squeeze and excitation (SE) mechanism of [66], and denoted *SE adapters*. This leads to the *multi-domain detector* of Figure 4.2 (b), where domain-specific SE adapters are introduced throughout the network to compensate for domain shift. On UODB, this detector outperforms that of Figure 4.2 (a) with $\sim$5 times fewer parameters.

In contrast, the *universal detector* of Figure 4.2 (c) shares all parameters/computations (other than output layers) across domains. It consists of a *single* network, which is always active.

This is the most efficient solution in terms of parameter sharing, but it is difficult for a single model to cover many domains with nontrivial domain shifts. Hence, this solution underperforms the multi-domain detector of Figure 4.2 (b). To overcome this problem, we propose the *domain-attentive universal detector* of Figure 4.2 (d). This leverages a novel domain attention (DA) module, in which a bank of the new universal SE adapters (active at all times) is first added, and a feature-based attention mechanism is then introduced to achieve domain sensitivity. This module learns to assign network activations to different domains, through the universal SE adapter bank, and soft-routs their responses by the domain-attention mechanism. This enables the adapters to specialize on individual domains. Since the process is data-driven, the number of domains does not have to match the number of datasets and datasets can span multiple domains. This allows the network to leverage shared knowledge across domains, which is not available in the common single-domain detectors. Our experiments, on the newly established UODB, show that this data-driven form of parameter/computation sharing enables substantially better multi-domain detection performance than the remaining architectures of Figure 4.2.

4.2 Related Work

The two stage detection framework of the R-CNN [51], Fast R-CNN [50] and Faster R-CNN [140] detectors has achieved great success in recent years. Many works have expanded this base architecture. For example, MS-CNN [10] and FPN [99] built a feature pyramid to effectively detect objects of various scales; the R-FCN [24] proposed a position-sensitive pooling to achieve further speed-ups; and the Cascade R-CNN [12] introduced a multi-stage cascade for high quality object detection. In parallel, single-stage object detectors, such as YOLO [137] and SSD [104], became popular for their fairly good performance and high speed. However, none of these detectors could reach high detection performance on more than one dataset/domain without finetuning. In the pre-deep learning era, [82] proposed a universal DPM [38] detector, by adding

dataset specific biases to the DPM. But this solution is limited since DPM is not comparable to deep learning detectors.

Multi-task learning (MTL) investigates how to jointly learn multiple tasks simultaneously, assuming a single input domain. Various multi-task networks [84, 188, 58, 102, 162, 195] have been proposed for joint solution of tasks such as object recognition, object detection, segmentation, edge detection, human pose, depth, action recognition, etc., by leveraging information sharing across tasks. However, the sharing is not always beneficial, sometimes hurting performance [37, 81]. To address this, [115] proposed a cross-stitch unit, which combines tasks of different types, eliminating the need to search through several architectures on a per task basis. [188] studied the common structure and relationships of several different tasks.

Multi-domain learning (MDL) addresses the learning of representations for multiple domains, known a priori [79, 120]. It uses a combination of parameters that are shared across domains and domain-specific parameters. The latter are adaptation parameters, inspired by works on domain adaptation [130, 108, 142, 110], where a model learned from a source domain is adapted to a target domain. [6] showed that multi-domain learning is feasible by simply adding domain-specific BN layers to an otherwise shared network. [135] learned multiple visual domains with residual adapters, while [136] empirically studied efficient parameterizations. However, they build on BN layers and are not suitable for detection, due to the batch constraints of detector training. Instead, we propose an alternative SE adapters, inspired by "Squeeze-and-Excitation" [66], to solve this problem.

[160] proposed a self-attention module for machine translation, and similarly, [163] proposed a non-local network for video classification, based on a spacetime dependency/attention mechanism. [66] focused on channel relationships, introducing the SE module to adaptatively recalibrate channel-wise feature responses, which achieved good results on ImageNet recognition. In this work, we introduce a domain attention module inspired by SE to make data-driven domain assignments of network activations, for the more challenging problem of universal object

detection.

4.3 Multi-Domain Object Detection

The problem of multi-domain object detection is to detect objects on various domains.

4.3.1 Universal Object Detection Benchmark

To train and evaluate universal/multi-domain object detection systems, we established a new universal object detection benchmark (UODB) of 11 datasets: Pascal VOC [36], WiderFace [182], KITTI [43], LISA [116], DOTA [168], COCO [98], Watercolor [72], Clipart [72], Comic [72], Kitchen [45] and DeepLesions [177]. This set includes the popular VOC [36] and COCO [98], composed of images of everyday objects, e.g. bikes, humans, animals, etc. The 20 VOC categories are replicated on CrossDomain [72] with three subsets of Watercolor, Clipart and Comic, with objects depicted in watercolor, clipart and comic styles, respectively. Kitchen [45] consists of common kitchen objects, collected with an hand-held Kinect, while WiderFace [182] contains human faces, collected on the web. Both KITTI [43] and LISA [116] depict traffic scenes, collected with cameras mounted on moving vehicles. KITTI covers the categories of vehicle, pedestrian and cyclist, while LISA is composed of traffic signs. DOTA [168] is a surveillance-style dataset, containing objects such as vehicles, planes, ships, harbors, etc. imaged from aerial cameras. Finally DeepLesion [177] is a dataset of lesions on medical CT images. A representative example of each dataset is shown in Figure 4.1. Some more details are summarized in Table 4.1. Altogether, UODB covers a wide range of variations in category, camera view, image style, etc, and thus establishes a good suite for the evaluation of universal/multi-domain object detection.

4.3.2 Single-Domain Detector Bank

The Faster R-CNN [140] is used as the baseline architecture of all detectors proposed in this work. As a single-domain object detector, the Faster R-CNN is implemented in two stages. First, a region proposal network (RPN) produces preliminary class-agnostic detection hypotheses. The second stage processes these with a region-of-interest detection network to output the final detections.

As illustrated in Figure 4.2 (a), the simplest solution to multi-domain detection is to use an independent detector per dataset. We use this detector bank as a multi-domain detection baseline. This solution is the most expensive, since it implies replicating all parameters of all detectors. Figure 4.3 shows the statistics (mean and variance) of the convolutional activations of the 11 detectors on the corresponding dataset. Some observations can be made. First, these statistics vary non-trivially across datasets. While the activation distributions of VOC and COCO are similar, DOTA, DeepLesion and CrossDomain have relatively different distributions. Second, the statistics vary across network layers. Early layers, which are more responsible for correcting domain shift, have more evident differences than latter layers. This tends to hold up to the output layers. These are responsible for the assignment of images to different categories and naturally differ. Interestingly, this behavior also holds for RPN layers, even though they are category-independent. Third, many layers have similar statistics across datasets. This is especially true for intermediate layers, suggesting that they can be shared by at least some domains.

4.3.3 Adaptive Multi-Domain Detector

Inspired by Figure 4.3, we propose an adaptive multi-domain detector, shown in Figure 4.2 (b). In this model, the output and RPN layers are domain-specific. The remainder of the network, e.g. all convolutional layers, is shared. However, to allow adaptation to new domains, we introduce some additional domain-specific layers, as is commonly done in MDL [135, 6].

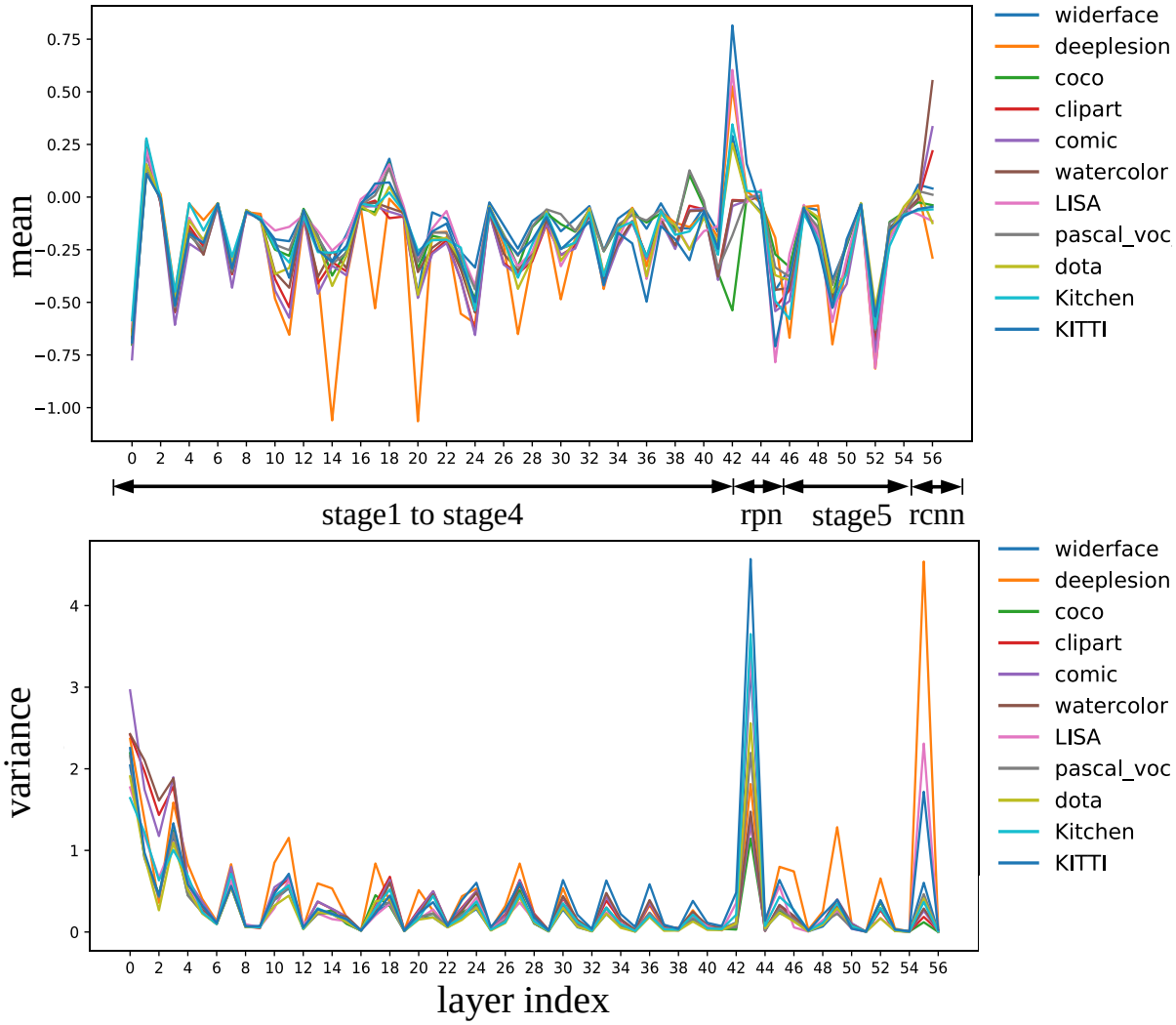
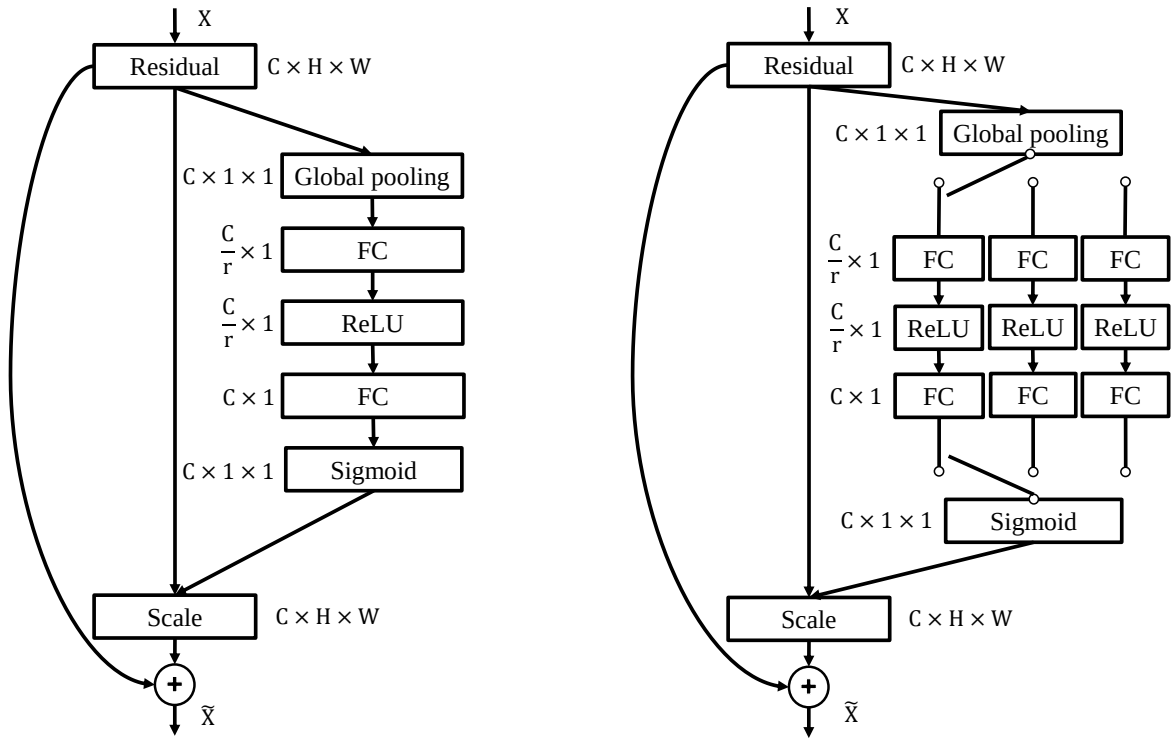


Figure 4.3: The activation statistics of all single-domain detectors.

These extra layers should be 1) sufficiently powerful to compensate for domain shift; 2) as light as possible to minimize parameters/computation. The adaptation layers of [135, 6] rely extensively on BN. This is unfeasible for detection, where BN layers have to be frozen, due to the small batch sizes allowable for detector training.

Instead, we have experimented with the squeeze-and-excitation (SE) module [66] of Figure 4.4 (a). There are a few reasons for this. First, feature-based attention is well known to be used in mammalian vision as a mechanism to adapt perception to different tasks and environments [185, 128, 165, 74, 184]. Hence, it seems natural to consider feature-based attention



(a) SE adapter (b) SE adapter bank

Figure 4.4: (a) block diagram of SE adapter and (b) SE adapter bank.

mechanisms for domain adaptation. Second, the SE is a module that accounts for interdependencies among channels to modulate channel responses. This can be seen as a a feature-based attention mechanism. Third the SE module has enabled the SENet to achieve state-of-the-art classification on ImageNet. Finally, it is a light-weight module. Even when added to each residual block of the ResNet [61] it increases the total parameter count by only $\sim$10%. This is close to what was reported by [135] for BN-based adapters. For all these reasons, we adopt the SE module as the atomic adaptation unit, used to build all domain adaptive detectors proposed in this work, and denote it by the *SE adapter*.

4.3.4 SE Adapters

Following [66], the *SE adapter* consists of the sequence of operations of Figure 4.4 (a): a global pooling layer, a fully connected (FC) layer, a ReLU layer, and a second FC layer,

implementing the computation

$$\mathbf{X}_{SE} = \mathbf{F}_{SE}(\mathbf{F}_{avg}(\mathbf{X})), \tag{4.1}$$

where $\mathbf{F}_{avg}$ is a global average pooling operator, and $\mathbf{F}_{SE}$ the combination of FC+ReLU+FC layers. The channel dimension reduction factor r, in Figure 4.4, is set as 16 in our experiments. To enable multi-domain object detection, the SE adapter is generalized to the architecture of Figure 4.4 (b), which is denoted as the *SE adapter bank.* This consists of adding a SE adapter branch per domain and a domain-switch, which allows the selection of the SE adapter associated with the domain of interest. Note that this architecture assumes this domain to be known a priori. It leads to the *multi-domain detector* of Figure 4.2 (b). Compared to Figure 4.2 (a), this model is up to 5 times smaller, while achieving better overall performance across the 11 datasets.

4.4 Domain-Universal Detection by Domain Attention

The detectors of the previous section require prior knowledge of the domain of interest. This is undesirable for autonomous systems, like robots or self-driving cars, where determining the domain is part of the problem to solve. In this section, we consider the design of *domain-universal detectors,* which eliminate this problem.

4.4.1 Simple Domain-Universal Detector

The simplest solution to universal detection, shown in Figure 4.2 (c), is to share a single detector by all tasks. Note that, even for this detector, the output layer has to be task-specific, by definition of the detection problem. This is not a problem because the task, namely what classes the system is trying to detect, is always known. Universality refers to the domain of input images that the detector processes, which does not have to be known in the case of Figure 4.2 (c). Beyond universal, the fully shared detector is the most efficient of all detectors considered in this work, as it has no domain-specific parameters. On the other hand, by forcing the same set

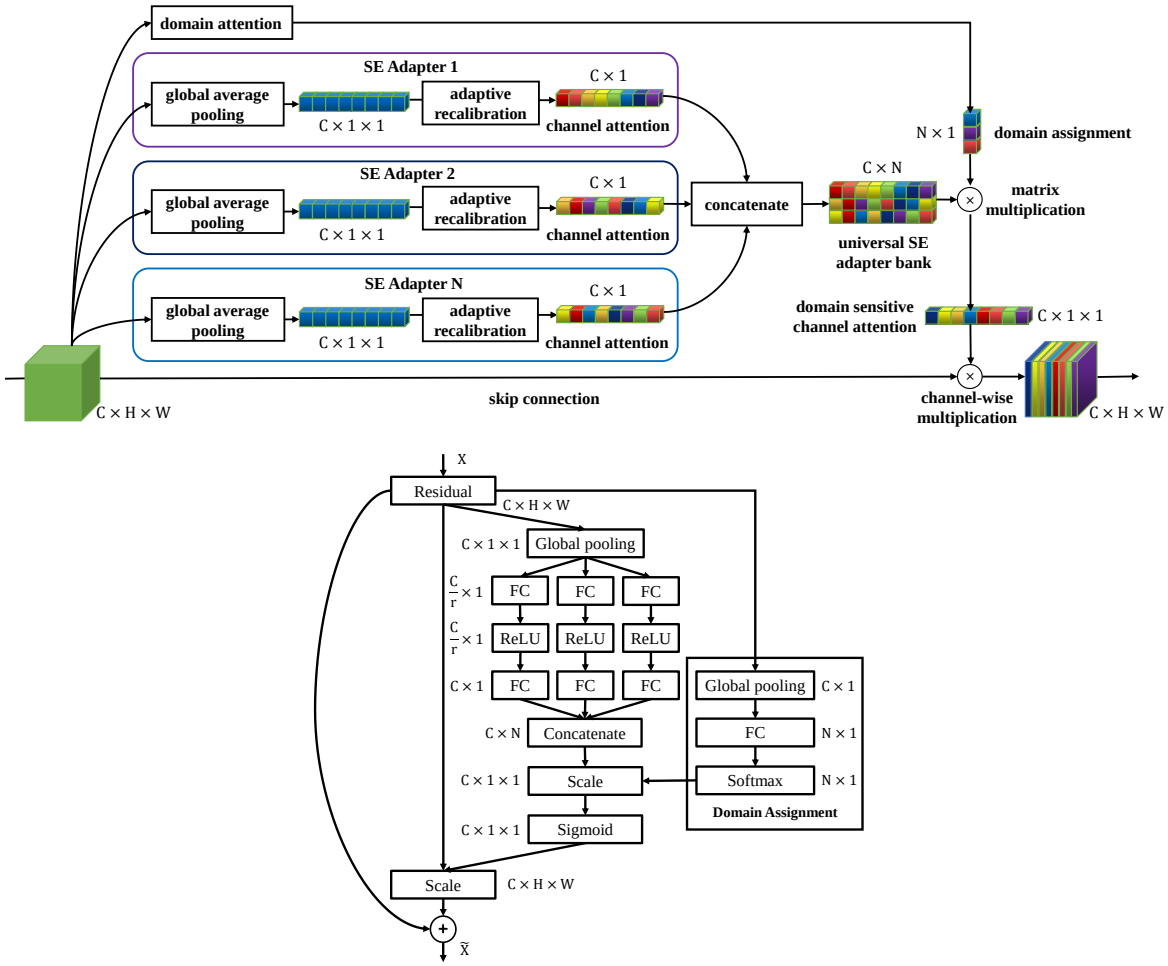


Figure 4.5: The block diagram (top) and the detailed view (down) of the proposed domain adaptation module.

of representations on all domains, it has little flexibility to deal with the statistical variations of Figure 4.3. In our experiments, this detector usually underperforms the multi-domain detectors of Figure 4.2 (a) and (b).

4.4.2 Domain-Attentive Universal Detector

Ideally, a universal detector should have some domain sensitivity, and be able to adapt to different domains. We also have found that there is also a benefit in using task-specific RPN layers, due to the observations of Figure 4.3. While this has a lot in common with multi-domain

detection, there are two main differences. First, the domain must be inferred automatically. Second, there is no need to tie domains and tasks. For example, the traffic tasks of Figure 4.1 operate on a common visual domain, "traffic scenes", which can have many sub-domains, e.g. due to weather conditions (sunny vs. rainy), environment (city vs. rural), etc. Depending on the specific operating conditions, any of the tasks may have to be solved in any of the domains. In fact, the domains may not even have clear semantics, i.e. they can be data-driven. In this case, there is no need to request that each detector operates on a single domain, and a soft domain-assignment makes more sense. Given all of this, while domain adaptation can still be implemented with the SE adapter of Figure 4.4 (a), the hard attention mechanism of Figure 4.4 (b), which forces the network to fully attend to a single domain, can be suboptimal. To address this limitations, we propose the domain adaptation (DA) module of Figure 4.5. This has two components, a *universal SE adapter bank* and a *domain attention* mechanism, which are discussed next.

4.4.3 Universal SE Adapter Bank

The universal SE (USE) Adapter Bank, shown in Figure 4.5, is an SE adapter bank similar to that of Figure 4.4 (b). The main difference is that there is no domain switching, i.e. the adapter bank is *universal.* This is implemented by concatenating the outputs of the individual domain adapters to form a universal representation space

$$\mathbf{X}_{USE} = [\mathbf{X}_{SE}^{1}, \mathbf{X}_{SE}^{2}, ..., \mathbf{X}_{SE}^{N}] \in \mathbb{R}^{C \times N}, \tag{4.2}$$

where N is the number of adapters and $\mathbf{X}_{SE}^{i}$ the output of each adapter, given by (4.1). Note that N is not necessarily identical to the number of detection tasks. The USE adapter bank can be seen as a non-linear generalization of the filter banks commonly used in signal processing [157]. Each branch (non-linearly) projects the input along a subspace matched to the statistics of a particular domain. The attention component then produces a domain-sensitive set of weights that are used

to combine these projections in a data-driven way. In this case, there is no need to know the operating domain in advance. In fact there may not even be a single domain, since an input image can excite multiple SE adapter branches.

4.4.4 Domain Attention

The attention component, of Figure 4.5, produces a domain-sensitive set of weights that are used to combine the SE bank projections. Motivated by the SE module, the domain attention component first applies a global pooling to the input feature map, to remove spatial dimensions, and then a softmax layer (linear layer plus softmax function)

$$\mathbf{S}_{DA} = \mathbf{F}_{DA}(\mathbf{X}) = \text{softmax}(\mathbf{W}_{DA}\mathbf{F}_{avg}(\mathbf{X})), \tag{4.3}$$

where $\mathbf{W}_{DA} \in \mathbb{R}^{N \times C}$ is the matrix of softmax layer weights. The vector $\mathbf{S}_{DA}$ is then used to weigh the USE bank output $\mathbf{X}_{USE}$, to produce a vector of domain adaptive responses

$$\mathbf{X}_{DA} = \mathbf{X}_{USE}\mathbf{S}_{DA} \in \mathbb{R}^{C \times 1}. \tag{4.4}$$

As in the SE module of [66], $\mathbf{X}_{DA}$ is finally used to channel-wise rescale the activations $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$ being adapted,

$$\widetilde{\mathbf{X}} = \mathbf{F}_{scale}(\mathbf{X}, \sigma(\mathbf{X}_{DA})) \tag{4.5}$$

where $\mathbf{F}_{scale}(\cdot)$ implements a channel-wise multiplication, and σ is the sigmoid function.

In this way, the USE bank captures the feature subspaces of the domains spanned by all datasets, and the DA mechanism soft-routes the USE projections. Both operations are data-driven, and operate with no prior knowledge of the domain. Unlike the hard attention mechanism of Figure 4.4 (b), this DA module enables information sharing across domains, leading to a more effective representation. In our experiments, the domain-attentive universal detector outperforms

Table 4.1: The dataset details, the domain-specific hyperparameters and the performance of the single-domain detectors. "T/V/T" means train/val/test, "size" the shortest side of inputs, *BS* RPN batch size, and S/R anchor "scales/aspect ratios".

dataset	dataset details			hyperparameters				mAP
	class	T/V/T	domain	size	*BS*	RoIs	S/R	
KITTI	3	7k/-/7k	traffic	576	256	128	12/3	64.3
WiderFace	1	13k/3k/16k	face	800	256	256	12/1	48.9
VOC	20	8k/8k/5k	natural	600	256	256	4/3	78.5
LISA	4	8k/-/2k	traffic	800	64	32	4/3	88.3
DOTA	15	14k/5k/10k	aerial	600	128	128	12/3	57.5
COCO	80	35k/5k/-	natural	800	256	256	4/3	47.3
Watercolor	6	1k/-/1k	watercolor	600	256	256	4/3	52.4
Clipart	6	0.5k/-/0.5k	clipart	600	256	256	4/3	32.1
Comic	20	1k/-/1k	comic	600	256	256	4/3	45.8
Kitchen	11	5k/-/2k	indoor	800	256	256	12/3	87.7
DeepLesion	1	23k/5k/5k	medical	512	128	64	12/3	51.3
Average	-	-	-	-	-	-	-	59.4

the other detectors of Figure 4.2.

4.5 Experiments

In all experiments, we used a PyTorch implementation [181] of the Faster R-CNN with the SE-ResNet-50[66]/ResNet50[61] pretrained on ImageNet, as the backbone for all detectors. Training started with a learning rate of 0.01 for 10 epochs and 0.001 for another 2 epochs on 8 synchronized GPUs, each holding 2 images per iteration. All samples of a batch are from a single (randomly sampled) dataset, and in each epoch, all samples of each dataset are processed only once. As is common for detection, the first convolutional layer, the first residual block and all BN layers are frozen, during training. These settings were used in all experiments, unless otherwise noted. Both multi-domain and universal detectors were trained on all domains of interest simultaneously.

The Faster R-CNN has many hyperparameters. In the literature, where detectors are tested on a single domain, these are tuned to the target dataset, for best performance. This is difficult,

and very tedious, to do over the 11 datasets now considered. We use the same hyperparameters across datasets, except when this is critical for performance and relatively easy to do, e.g. the choice of anchors. The main dataset-specific hyperparameters are shown in Table 4.1.

4.5.1 Datasets and Evaluation

Our experiments used the new UODB benchmark introduced in Section 4.3.1. For Watercolor [72], Clipart [72], Comic [72], Kitchen [45] and DeepLesion [177], we trained on the official `trainval` sets and tested on the `test` set. For Pascal VOC [36], we trained on VOC2007 and VOC2012 `trainval` set and tested on VOC2007 `test` set. For WiderFace [182], we trained on the `train` set and tested on the `val` set. For KITTI [43], we followed the train/val splitting of [10] for development and trained on the `trainval` set for the final results on `test` set. For LISA [116], we trained on the `train` set and tested on the `val` set. For DOTA [168], we followed the pre-processing of [168], trained on `train` set and tested on `val` set. For MS-COCO [98], we trained on COCO 2014 `valminusminival` and tested on `minival`, to shorten the experimental period.

All detectors were evaluated on each dataset individually. The Pascal VOC mean average precision (mAP) was used for evaluation in all cases. The average mAPs was used as the overall measure of universal/multi-domain detection performance. The domain attentive universal detector was also evaluated using the official evaluation tool of each dataset, for comparison with the literature.

4.5.2 Single-Domain Detection

Table 4.1 shows the results of the single-domain detector bank of Figure 4.2 (a) on all datasets. Our VOC baseline with the SE-ResNet-50 is 78.5, and better than the Faster R-CNN performance of [139, 61] (76.4 mAP for ResNet-101). The other entries in the table are

Table 4.2: The comparison on multi-domain detection. † denotes fixed assignment. "time" is the relatively run-times on the five datasets when the domain is unknown.

	Params	time	KITTI	VOC	WiderFace	LISA	Kitchen	Avg
single-domain	155.1M	5x	64.3	78.5	48.8	88.3	87.7	73.5
adaptive	42.37M	6x	67.8	78.9	49.9	**88.5**	86.0	74.2
BNA [6]	31.72M	5x	64.0	71.9	44.0	66.8	84.3	66.2
RA [135]	82.72M	6x	64.3	70.5	46.9	69.1	84.6	67.1
universal	29.58M	1x	64.2	75.0	43.5	88.9	86.8	71.5
universal+DA†	37.47M	1.3x	67.5	79.0	49.8	88.2	88.0	74.6
universal+DA	37.56M	1.33x	**67.9**	**79.2**	**52.2**	87.5	**88.5**	**75.1**

Table 4.3: The effect of SE adapters number.

# adapters	Params	KITTI	VOC	WiderFace	LISA	Kitchen	Avg
single	155.1M	64.3	78.5	48.8	88.3	87.7	73.5
3	34.82M	67.6	78.0	51.9	88.1	87.1	74.5
5	37.88M	**67.9**	79.2	**52.2**	87.5	88.5	75.1
7	40.94M	**67.9**	**79.6**	**52.2**	**89.5**	**88.7**	**75.6**
9	44.01M	67.7	79.2	**52.2**	**89.5**	86.9	75.1

incomparable to the literature, where different evaluation metrics/tools are used for different datasets. The detector bank is a fairly strong baseline for multi-domain detection (average mAP of 59.4).

4.5.3 Multi-Domain Detection

Table 4.2 compares the multi-domain object detection performance of all architectures of Figure 4.2. For simplicity, only five datasets (VOC, KITTI, WiderFace, LISA and Kitchen) were used in this section. The table confirms that the adaptive multi-domain detector of Section 4.3.3 ("adaptive") is light-weight, only adding $\sim$11M parameters to the Faster R-CNN over the five datasets. Nevertheless, it outperforms the much more expensive single-domain detector bank by 0.7 points. Note that the latter is a strong baseline, showing the multi-domain detector can beat individually trained models with a fraction of the computation. Table 4.2 also shows that the proposed SE adapter significantly outperforms the BN adapter (BNA) of [6] and the residual adapter (RA) or [135], previously proposed for classification. This is not surprising, given the

above discussed inadequacy of BN as an adaptation mechanism for object detection.

The universal detector of Figure 4.2 (c) is even more efficient, adding only 0.5M parameters to the Faster R-CNN, accounting for domain-specific RPN and output layers. However, its performance ("universal" in Table 4.2) is much weaker than that of the adaptive multi-domain detector (1.7 points). Finally, the domain-attentive universal detector ("universal+DA") has the best performance. With a $\sim$7% parameter increase per domain, i.e. comparable to the multi-domain detector, it outperforms the single-domain bank baseline by 1.6 points. To assess the importance of data-driven domain attention mechanism of Figure 4.5 (b), we fixed the soft domain assignments, simply averaging the SE adapter responses, during both training and inference. This (denoted "universal+DA†") caused a performance drop of 0.5 point. Finally, Table 4.2 shows the relative run-times of all methods on the five datasets, when the domain is unknown. It can be seen that "universal+DA" is about 4$\times$ faster than the multi-domain detectors ("single-domain" and "adaptive") and only 1.33$\times$ slower than "universal".

4.5.4 Effect of the number of SE adapters

For the USE bank of Figure 4.5 (b), the number N of SE adapters does not have to match the number of detection tasks. Table 4.3 summarizes how the performance of the domain attentive universal detector depends on N. For simplicity, we again use 5 datasets in this experiment. As we can see in Table 4.3, more SE adapters will not always lead to better performance. Generally, performance improves with the number of adapters until we reached 7 adapters, adding 9 adapters will only get similar performance with 5 adapters and increase parameters number a lot. On the other hand, the number of parameters increases linearly with the number of adapters. In these experiments, the best trade-off between performance and parameters is around 5 adapters. This suggests that, while a good rule of thumb is to use "as many adapters as domains", fewer adapters can be used when complexity is at a premium.

Table 4.4: Overall results on the full universal object detection benchmark (11 datasets). "ada" indicates adapter, "WF" WiderFace, "DL" DeepLesion, "WC" Watercolor.

	# ada	Params	DA index	KITTI	VOC	WF	LISA	Kitchen	COCO	DOTA	DL	Comic	Clipart	WC	Avg
single-domain	-	340.7M	-	64.3	78.5	48.8	88.3	87.7	**47.3**	**57.5**	51.2	45.8	32.1	52.6	59.4
adaptive	11	58.13M	-	68.0	82.1	50.6	**88.5**	87.2	45.7	54.1	53.0	50.0	56.1	57.8	63.0
universal	-	32.60M	-	67.5	80.9	45.5	87.1	88.5	45.5	54.7	45.3	51.1	43.1	47.0	59.7
universal+DA	11	58.29M	all	**68.1**	82.0	51.6	88.3	**90.1**	46.5	57.0	**57.3**	50.7	53.1	58.4	63.8
universal+DA*	6	41.74M	first+middle	67.6	**82.7**	**51.8**	87.9	88.7	46.8	57.0	54.8	52.6	54.6	58.2	63.9
universal+DA*	8	44.80M	first+middle	68.0	82.4	51.3	87.6	90.0	47.0	56.3	53.4	**53.4**	**55.8**	**60.6**	**64.2**

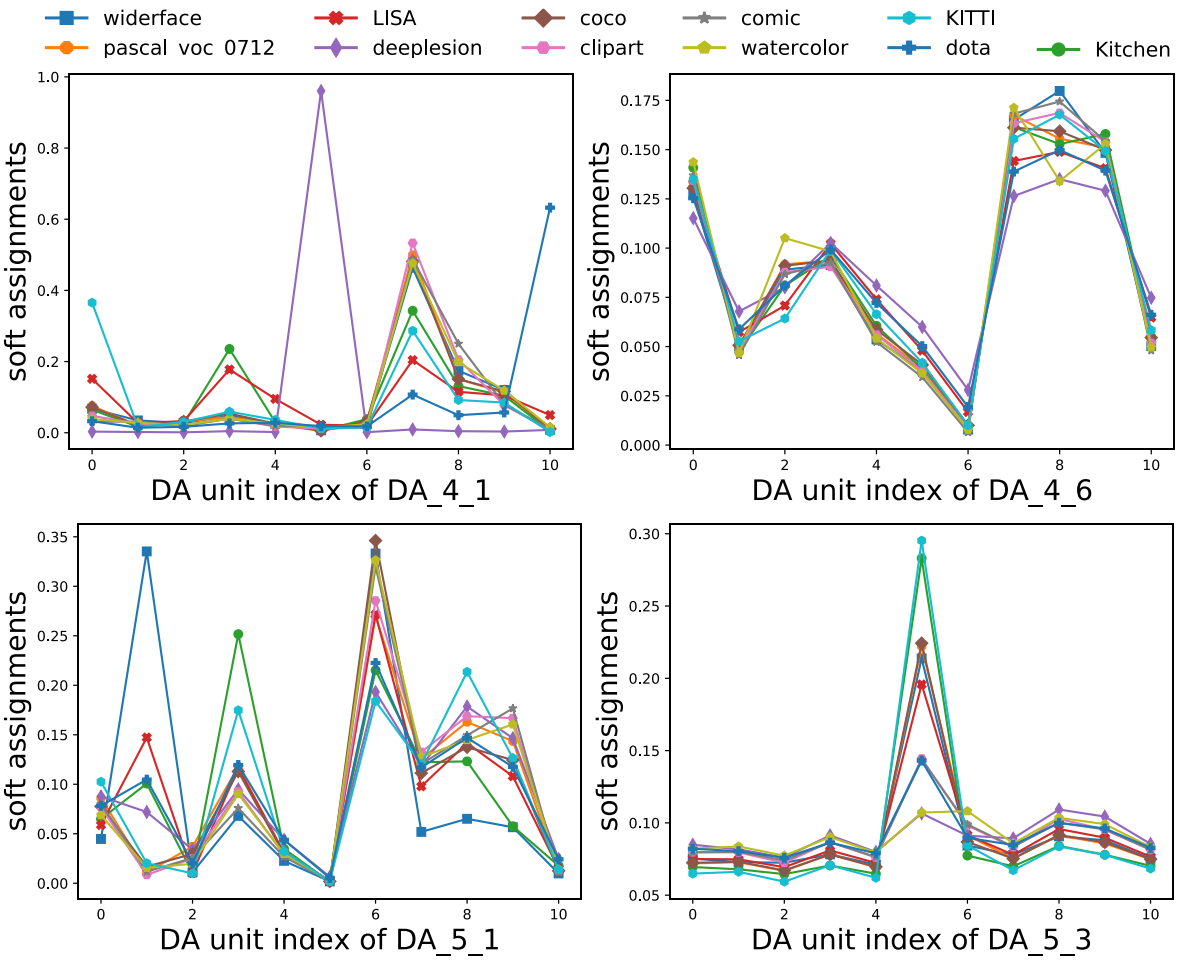


Figure 4.6: Soft assignments across SE units for all datasets.

4.5.5 Results on the full benchmark

Table 4.4 presents results on the full benchmark. The settings are as above, but we used 10 epochs with learning rate 0.1, and then 4 epochs with 0.01 on 8 GPUs, each holding 2 images. For universal detector without our proposed module, training with learning rate 0.1, even with warming up for 1 epoch, will always get loss exploding, therefore, we will use $0.1\times$lr, i.e. 0.01, to train universal detector. The universal detector performs comparably to the single-domain detector bank, with 10 times fewer parameters. The domain-attentive universal detector ("universal+DA") improves baseline performance by 4.4 points with a 5-fold parameter decrease. It has large performance gains (>5 points) on DeepLesion, Comic, and Clipart. This is because

Comic/Clipart contain underpopulated classes, greatly benefiting from information leveraged from other domains. The large gain of DeepLesion is quite interesting, given the nontrivial domain shift between its medical CT images and the RGB images of the other datasets. The gains are mild for VOC, KITTI, Kitchen, WiderFace and WaterColor (1$\sim$5 points), and none for COCO, LISA and DOTA. In contrast, for the universal detector, joint training is not always beneficial. This shows the importance of domain sensitivity for universal detection.

To investigate what was learned by the domain attention module of Figure 4.5 (b), we show the soft assignments of each dataset, averaged over its validation set, in Figure 4.6. Only the first and last blocks of the 4th and 5th residual stages are shown. The fact that some datasets, e.g. VOC and COCO, have very similar assignment distributions, suggests a substantial domain overlap. On the other hand, DOTA and DeepLesion have distributions quite distinct from the remaining. For example, on block "DA_4_1", DeepLesion fully occupies a single domain. These observations are consistent with Figure 4.3, indicating that the proposed DA module is able to learn domain-specific knowledge.

A comparison of the first and the last blocks of each residual stage, e.g. "DA_4_1" v.s. "DA_4_6", shows that the latter are much less domain sensitive than the former, suggesting that they could be made universal. To test this hypothesis, we trained a model with only 6 SE adapters for the 11 datasets, and only in the first and middle blocks, e.g. "DA_4_1" and "DA_4_3", all the other blocks will only add 1 SE adapter without domain assignment module. This model, "universal+DA*", achieved the best performance with much less parameters than the "universal+DA" detector of 11 adapters. It outperformed the single domain baseline by 4.5 points. We also trained a model with 8 SE adapters, adding another two adapters will get another 0.3 points increase, outperforms single-domain baseline by 4.8 points.

Table 4.5: The comparison on VOC 2007 `test`. ‡/† denotes with COCO `trainval`/`val`

	Backbone	mAP
Faster-RCNN [140]	ResNet-101	76.4
R-FCN [24]	ResNet-50	77.0
Faster-RCNN‡ [139]	VGG16	78.8
Faster-RCNN (ours)	ResNet-50	78.0
Faster-RCNN (ours)	SE-ResNet-50	78.5
Faster-RCNN+DA	DA-ResNet-50	79.6
Faster-RCNN+DA†	DA-ResNet-50	82.7

Table 4.6: The comparison on WiderFace `Val`.

	Backbone	Easy	Medium	Hard
Faster-RCNN [140]	VGG-16	0.907	0.850	0.492
MS-CNN [10]	VGG-16	0.916	0.903	0.802
HR [67]	ResNet-101	0.925	0.910	0.806
SSH [119]	VGG-16	0.931	0.921	0.845
Faster-RCNN (ours)	ResNet-50	0.905	0.864	0.548
Faster-RCNN (ours)	SE-ResNet-50	0.910	0.872	0.556
Faster-RCNN+DA	DA-ResNet-50	0.914	0.882	0.587

4.5.6 Official Evaluation

Since, to the best of our knowledge, this is the first work to explore universal/multi-domain object detection on 11 datasets, there is no literature for a direct comparison. Instead, we compared the "universal+DA*" detector of Table 4.4 to the literature using the official evaluation for each dataset. This is an unfair comparison, since the universal detector has to remember 11 tasks. All single-domain detector will use SE-ResNet50 as backbone. On VOC, we trained two models, with/without COCO. Results are shown in Table 4.5, where all methods were trained on Pascal VOC 07+12 `trainval`. Note that our Faster R-CNN baseline (SE-ResNet-50 backbone) is stronger than that of [61] (ResNet-101). Adding universal domain adapters improved on the baseline by more than 1.1 points. Adding COCO enabled another 3.1 points. Note that, 1) this universal training is different from the training scheme of [139] (the network trained on COCO then finetuned on VOC), where the final model is only optimized for VOC; and 2) only the 35k images of COCO2014 `valminusminival` were used.

Table 4.7: The comparison on KITTI `test` set of car.

	Backbone	Moderate	Easy	Hard
Faster-RCNN [140]	VGG-16	81.84	86.71	71.12
SDP+CRC [180]	VGG-16	83.53	90.33	71.13
YOLOv3 [138]	Darknet-53	84.13	84.30	76.34
MS-CNN [10]	VGG-16	88.83	90.46	74.76
F-PointNet [133]	PointNet	90.00	90.78	80.80
Faster-RCNN (ours)	ResNet-50	80.28	90.43	70.9
Faster-RCNN (ours)	SE-ResNet-50	81.83	90.34	71.23
Faster-RCNN+DA	DA-ResNet-50	88.23	90.45	74.21

Table 4.8: Sensitivity at 4 FPs per image on DeepLesion `test` set.

	Backbone	Sensitivity
Faster-RCNN [140]	VGG-16	81.62
R-FCN [24]	VGG-16	82.21
3-DCE, 9 Slices [176]	VGG-16	84.34
3-DCE, 27 Slices [176]	VGG-16	85.65
Faster-RCNN (ours)	ResNet-50	81.34
Faster-RCNN (ours)	SE-ResNet-50	82.44
Faster-RCNN+DA	DA-ResNet-50	87.29

The baseline was the default Faster R-CNN that initially worked on VOC, with minimum dataset-specific changes, e.g. in Table 4.1. Table 4.7 shows that this performed weakly on KITTI. However, the addition of adapters, enabled a gain of 6.4 points (`Moderate` setting). This is comparable to detectors optimized explicitly on KITTI, e.g. MS-CNN [10] and F-PointNet [133]. For WiderFace, which has enough training face instances, the gains of shared knowledge are smaller (see Table 4.6). On the other hand, on DeepLesion and CrossDomain (Clipart, Comic and Watercolor), see Table 4.8 and 4.9 respectively, the domain attentive universal detector significantly outperformed the state-of-the-art. Although DeepLesion contains more than 33k images, most of them only contain one lesion, which will make detector unable to get adequate training. For Cross-domain datasets, they only contain a few thousands samples. Single-domain detector is easy to be over-fitting when fine-tuned on so small datasets. Jointly training with other datasets, especially with VOC and MS-COCO which share some categories with cross-domain datasets, will benefit the representation learning process a lot for Cross-domain datasets and

Table 4.9: The comparison on Clipart, Watercolor and Comic `test` set.

	Backbone	Clipart	Watercolor	Comic
ADDA [155]	VGG-16	27.4	49.8	49.8
Faster-RCNN[140]	VGG-16	26.2	-	-
SSD300 [104]	VGG-16	26.8	49.6	24.9
Faster-RCNN+DT+PL[72]	VGG-16	34.9	-	-
SSD300+DT+PL[72]	VGG-16	46.0	54.3	37.2
Faster-RCNN (ours)	ResNet-50	36.7	51.2	42.6
Faster-RCNN (ours)	SE-ResNet-50	32.1	52.6	45.8
Faster-RCNN+DA	DA-ResNet-50	54.6	58.2	52.6

mitigate over-fitting problem. Overall, these results show that a single detector, which operates on 11 datasets, is competitive with single-domain detectors in highly researched datasets, such as VOC or KITTI, and substantially better than the state-of-the-art in less explored domains. This is achieved with a relatively minor increase in parameters, vastly smaller than that needed to deploy 11 single task detectors.

4.6 Conclusion

We have investigated the unexplored and challenging problem of universal/multi-domain object detection. We proposed a universal detector that requires no prior domain knowledge, consisting of a *single* network that is active for all tasks. The proposed detector achieves domain sensitivity through a novel data-driven domain adaptation module and was shown to outperform multiple universal/multi-domain detectors on a newly established benchmark, and even individual detectors optimized for a single task.

4.7 Acknowledgements

Chapter 4 is, in full, based on the material as it appears in the publication of “Towards Universal Object Detection by Domain Attention”, Xudong Wang, Zhaowei Cai, Dashan Gao,

and Nuno Vasconcelos, In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019. The dissertation author was a joint primary investigator and author of this material.

Chapter 5

Learning Complexity-Aware Cascades

5.1 Introduction

Pedestrian detection is an important problem in computer vision, with application to smart vehicles, surveillance, etc. Due to the real-time implementation requirements of many of these applications, the detector cascade architecture of [161] has a long history in pedestrian detection. It relies on the sliding window paradigm, processing each image patch with a sequence of binary classifiers, known as cascade stages. Each stage either rejects the image patch or passes it to the subsequent stages. This leads to two important properties. First, because most patches can be rejected by early stages of low complexity, detection can be fast. Second, because the remaining false positives can be rejected by complex detectors, deeper into the cascade, it can also be accurate. In fact, because final stages are rarely used, their complexity is not an impediment to fast detection.

While the cascade detection principle is intuitive, its implementation is far from trivial. Early cascade designs required extensive heuristics to determine the configuration of cascade stages [161, 171, 8]. A common assumption, by these methods, is that all features have equivalent complexity. This significantly simplifies the design, which reduces to choosing features so as to maximize detection accuracy. In fact, popular methods [8, 29] simply use a boosting algorithm (typically AdaBoost [40]) to design a non-cascaded classifier and then transform it into a cascade, by addition of thresholds. These approaches suffer from the problem that the "equivalent feature complexity" hypothesis only produces sensible cascades when applied to features that indeed have similar complexity.

In result, these methods have difficulties to accommodate features of widely varying complexity. This is a problem for applications that require diverse feature sets [4, 127, 192], and has become critical after the introduction of deep learning [86, 150]. On one hand, it is now well known that high detection rates require deep learning models. On the other, it is quite difficult to learn a cascade that combines these with the very efficient detectors needed in the early cascade

stages. In fact, a complexity insensitive boosting algorithm, such as AdaBoost, will start by selecting the deep features, which are most accurate but also the most complex, for the early stages. This produces very slow cascades.

In object detection, an alternative strategy has emerged to address the intractability of sliding windows for deep learning. This consists of using an object proposal mechanism [158], which selects patches to process by a more powerful classifier. Various object detectors, such as the R-CNN [51], Fast-RCNN [50] and Faster-RCNN [140], are based on this architecture. While efficient, this strategy effectively implements a two stage cascade. Since there is no reason to believe that two stages guarantee the optimal trade-off between accuracy and speed, it does not necessarily guarantee the highest detection rate for a given complexity. For pedestrian detection, object proposals are frequently implemented with weak pedestrian detectors, sometimes cascaded detectors themselves [64]. The success of the somewhat arbitrary decomposition into a cascade and a deep classifier suggests that good performance should be possible for cascades with stages of deep learning models.

The introduction of deep learning has also generated a shift in the computer architectures used for object recognition. While classical cascades can be implemented on CPUs, deep learning model require dedicated GPUs. These create problems to applications, due to their high cost and energy consumption. In practice, detector design requires consideration of the trade-offs between accuracy, running time, energy consumption, and implementation cost. These different facets of "complexity" vary in importance from application to application. While speed is critical for a smart car braking system, energy (battery time) is much more important for drone applications. On platforms like drones or home surveillance, it may not be cost-effective to use a GPU-based detector of high accuracy, e.g it could be better to eliminate the GPU and complement a classic CPU-based detector with one using radar or some other modality.

In a CPU architecture, all these costs are (approximately) linear on the number of computations required for the detection. Computation (or speed) can thus be used as a universal measure

of complexity, simplifying the design of the loss functions needed to learn complexity-aware algorithms. For GPU-based detectors this is much harder to accomplish. In this case, computation, speed, and energy consumption are not linearly related, due to the parallel implementation of all computations. In the detection context, where different image areas have different classification complexity, the optimal allocation of computation and energy is spatially varying. This is, however, at odds with the parallel implementation of image-wide layers, which guarantees optimal speed. Modern convolutional neural network (CNN) libraries favor the latter, leading to detectors that are fast but perform many unnecessary computations, increasing detector cost and energy consumption.

In this work, we address these problems by seeking an algorithm for optimal cascade learning under a criterion that penalizes *both* detection errors and complexity. For the latter, we introduce a definition of *complexity risk* akin to the empirical risk commonly used for classifier design. This makes it possible to define quantities such as complexity margins and complexity losses, and account for these in the learning process. We do this with recourse to a Lagrangian formulation, which optimizes for the usual classification risk under a constraint in the complexity risk. While the Lagrangian formulation can account for any of the facets of complexity discussed above, we focus on computational complexity in this work. A boosting algorithm that minimizes this Lagrangian is then derived.

This algorithm, denoted *Complexity-Aware Cascade Training* (CompACT), is shown to select inexpensive features in the early cascade stages, pushing the more expensive ones to the later stages. This enables the combination of features of vastly different complexities in a single detector. These properties are demonstrated by the successful application of CompACT to the problem of pedestrian detection, using a pool of features ranging from Haar wavelets to deep CNNs. In particular, we show that it is possible to embed deep CNNs as the final stages of classical detector cascades. This leads to mixed CPU/GPU solutions, that provide practitioners with flexible families of models, ranging from cheap and energy efficient CPU-based cascades, to

cascades using GPU stages that emphasize speed or recognition accuracy.

Overall, this work makes four major contributions. First, it proposes a novel algorithm for learning complexity aware cascades, which optimally trade-off accuracy and speed. To the best of our knowledge, this is the first algorithm to *explicitly* account for variable feature complexity in cascade learning, supporting weak learners of widely different complexities. Second, CompACT seamlessly integrates handcrafted and CNN features in a unified cascaded detector. This generalizes the object proposal architecture, enabling the integration of CNN stages with stages of any other complexity. Third, it is shown that many large and expensive CNN models can be optimally embedded into the proposed CompACT cascades, enabling a range of complexities and accuracies. Finally, it is shown that the embedding of a large CNN detector, the MS-CNN [10], introduced in Chapter 2, enables CompACT cascades to achieve accurate pedestrian detection rates on Caltech [32] and KITTI [43], at fairly fast speeds. While these contributions only address the design of cascaded detectors by boosting, we hope that the ideas will inspire subsequent research on explicit modeling of complexity into the objective functions used to train neural networks end-to-end.

5.2 Related Work

Cascades have long been used to detect objects such as faces [161, 8, 171, 170], pedestrians [29, 144], or cars [145]. Early approaches used heuristics to find a cascade configuration of good trade-off between accuracy and complexity [161, 8, 171, 170]. More recently, there have been efforts to optimize this trade-off [112, 144, 145, 196]. For example, [196] added a complexity term to the RealBoost loss, and [112, 144, 145] introduced the Lagrangian formulation we adopt. However, these methods use a single feature family throughout the cascade. The need for early cascades stages to be very efficient restricts this to a simple feature, e.g. a decision stump. For pedestrian detection, this is usually applied to the integral channel features of [30].

These extend the Haar-like features of [161] into a set of color and histogram-of-gradients (HOG) channels. A computationally efficient version of [161], denoted the aggregate channel features (ACF), was introduced in [29].

More recently, [127] complemented ACF with local binary patterns (LBP) and covariance features, for better detection accuracy. Several works proposed alternative feature channels, obtained by convolving different filters with the original HOG+LUV channels [191, 192, 4, 121]. The SquaresChnFtrs of [4] reduce the large number of features of [30, 161] to 16 box-like filters of various sizes. [121] extended the locally decorrelated features of [56] to ACF, learning four 5×5 PCA-like filters from each of the ACF channels. Instead of empirical filter design, [191] exploited prior knowledge about pedestrian shape to design informed filters. They later found, however, that such filters are actually not needed [192]. Instead, the number of filters appears to be the most important variable: features as simple as checkerboard patterns, or purely random filters, can achieve very good performance, as long as there are enough of them. However, these methods are relatively slow, since good performance requires convolution with large numbers of filters [127, 192].

While deep convolutional classifiers achieve impressive results on general object detection [51], e.g. on VOC, COCO or ImageNet, initial attempts with deep learning did not excel at pedestrian detection [147, 124, 5]. The difficulty of sliding window implementation of deep models motivated the use of object proposal mechanisms [51, 164, 64] that pre-select the most promising image patches. Early models, such as the R-CNN [51] were fairly slow, but the introduction of the Fast-RCNN [50], allowing the computation of CNN features once per image, increased detection speed by an order of magnitude. Nevertheless, this model required an independent bottom-up stage for proposal generation. Later, the Faster-RCNN [140] integrated the generation of object proposals and region-wise classification within a single neural network, leading to a significant speedup.

Following these works, [178, 190, 92] presented good results for pedestrian detection.

However, large CNN models tend to be costly and consume large amounts of energy, due to their GPU requirements. The massively parallel implementation required for speed makes difficult the optimal allocation of computation to different image areas. This makes it difficult to optimize the trade-offs between accuracy and speed, accuracy and energy consumption, accuracy and cost, etc. For example, [137, 104] obtained significantly higher speeds than those of the Faster-RCNN with a more efficient one-shot detection architecture but at the cost of substantially weaker detection rates. The implementation of high accuracy GPU-based detectors with low energy consumption or low-cost is mostly open at this point.

In this work, we define the optimal detector as that of maximal detection accuracy under a complexity constraint. Under this definition, the two-stage (proposal plus classification) cascade is not necessarily optimal. A more general multi-stage cascade could, in principle, achieve a better trade-off between detection accuracy and speed. This motivated us to consider multi-stage cascades that 1) include deep learning stages and 2) seek the optimal trade-off between accuracy and complexity. This is accomplished with a generalization of the classic boosting framework for cascade design, which complements the boosting loss with a complexity penalty.

The addition of this complexity term also establishes an explicit connection between complexity constrained detection and regularization. It is well known that AdaBoost can overfit when either the number of weak learners or their complexity are unchecked. In general, overfitting propensity depends on the VC-dimension or Rademacher complexity of the detector [146]. This has motivated the inclusion of complexity measures, e.g. Rademacher complexity [18], into the boosting loss. CompACT is similar in that it considers complexity albeit computation, not Rademacher complexity. Note, however, that both complexity measures are monotonically increasing functions of decision tree depth. Our experiments show that the complexity constraint in the proposed CompACT can indeed improve resistance to overfitting.

5.3 Complexity-Aware Cascade

In this section we introduce the CompACT algorithm.

5.3.1 AdaBoost

A decision rule $h(x) = sign[F(x)]$ of predictor $F(x)$ maps a feature vector $x \in \mathcal{X}$ to a class label $y \in \mathcal{Y} = \{-1, 1\}$. Boosting learns a strong decision rule by combining a set of weak learners $f_k(x)$,

$$F(x) = \sum_k f_k(x), \tag{5.1}$$

by functional gradient descent on a classification risk [41, 114]. AdaBoost [40] uses the exponential loss $\phi(yF(x)) = e^{-yF(x)}$, minimizing the empirical risk

$$\mathcal{R}_E[F] \simeq \frac{1}{|S_t|} \sum_i e^{-y_i F(x_i)}, \tag{5.2}$$

on a training set $S_t = \{(x_i, y_i)\}$. Boosting iterations compute the functional derivative of (5.2) along the direction of weak learner $g(x)$ at the current predictor $F(x)$,

$$\begin{aligned} < \delta\mathcal{R}_E[F], g > &= \frac{d}{d\varepsilon} \mathcal{R}_E[F + \varepsilon g]\big|_{\varepsilon=0} \\ &= \frac{1}{|S_t|} \sum_i \left[\frac{d}{d\varepsilon} e^{-y_i(F(x_i)+\varepsilon g(x_i))} \right] \Big|_{\varepsilon=0} \\ &= -\frac{1}{|S_t|} \sum_i y_i w_i g(x_i), \end{aligned} \tag{5.3}$$

where $w_i = w(y_i, x_i) = e^{-y_i F(x_i)}$. The predictor is updated by selecting the steepest descent direction within a weak learner pool $\mathbf{G} = \{g_1(x), \cdots, g_n(x)\}$,

$$\begin{aligned} g^*(x) &= \arg\max_{g \in \mathbf{G}} < -\delta \mathcal{R}_E[F], g > \\ &= \arg\max_{g \in \mathbf{G}} \frac{1}{|S_t|} \sum_i y_i w_i g(x_i). \end{aligned} \tag{5.4}$$

The optimal step size for the update is

$$\alpha^* = \arg\min_{\alpha} \mathcal{R}_E[F + \alpha g^*]. \tag{5.5}$$

For binary $g^*(x)$, this has a closed form solution

$$\alpha^* = \frac{1}{2} \log \frac{\sum_{i|y_i = g^*(x)} w_i^k}{\sum_{i|y_i \neq g^*(x)} w_i^k}. \tag{5.6}$$

Otherwise, the optimal step size is found by a line search.

5.3.2 Complexity-Aware Learning

Complexity-aware learning seeks the best trade-off between classification accuracy and complexity. This is a constrained optimization problem, where classification risk is minimized under a bound on a complexity risk $R_C[F]$,

$$F^*(x) = \arg\min_F R_E[F] \quad s.t. \quad R_C[F] < \gamma, \tag{5.7}$$

which is solved by minimizing the Lagrangian

$$\mathcal{L}[F] = \mathcal{R}_E[F] + \eta \mathcal{R}_C[F], \tag{5.8}$$

where η is a Lagrange multiplier that only depends on γ. To define a complexity risk, we note that (5.2) can be written as

$$\mathcal{R}_E[F] \simeq \frac{1}{|S_t|}\sum_i \phi[\xi(y_i, F(x_i))], \tag{5.9}$$

with $\phi(v) = e^{-v}$ and $\xi(y, F(x)) = yF(x)$. The function $\xi(\cdot)$ is the margin of example x under predictor $F(\cdot)$ and measures the confidence of the classification. Large positive (negative) margins indicate that x is correctly (incorrectly) classified with high confidence, and the margin is zero for examples on the boundary. The loss $\phi(\cdot)$ is usually monotonically decreasing, penalizing all examples with less than a small positive margin. This forces the learning algorithm to concentrate on these examples, producing as few negative margins as possible. The exponential loss of AdaBoost makes the penalty exponential on the confidence of incorrectly classified examples.

In this work, we adopt a complexity risk of similar form

$$\mathcal{R}_C[F] \simeq \frac{1}{|S_t|}\sum_i \tau[\kappa(y_i, F(x_i))], \tag{5.10}$$

where $\kappa[y, F(x)]$ measures the complexity of classifying example x with $F(\cdot)$ and $\tau(\cdot)$ is a non-negative complexity loss. Drawing inspiration from the classification risk, we measure complexity with the complexity margin

$$\kappa[y, F(x)] = y\Omega(F(x)). \tag{5.11}$$

While $\Omega(F(x))$ can be any measure of complexity, in this work we focus on computational complexity, setting $\Omega(F(x))$ to the number of operations required to evaluate $F(x)$. (5.11) assigns positive (negative) complexity to positive (negative) examples, reflecting the fact that computation spent on negative examples is "wasted" or "negative" while that spent on positives is "justified" or "positive". While positives have to survive all cascade stages, negatives should be rejected with little computation. The loss $\tau(v)$ then determines the complexity-aware behavior of learning algorithms. For example, a decreasing $\tau(v)$ for $v < 0$ penalizes negative examples of large

complexity. This encourages classifiers that reject negatives with as little computation as possible. On the other hand, an increasing $\tau(v)$ for $v > 0$ penalizes positives of large complexity.

5.3.3 Embedded Cascade

A cascaded classifier is a sequence of classification stages $h_i(x) = sgn[F_i(x) + T_i]$, where T_i is a threshold. A popular architecture, which we adopt in this work, is the embedded cascade, whose predictor has the embedded structure

$$F_k(x) = F_{k-1}(x) + f_k(x) = \sum_{j=1}^{k} f_j(x). \tag{5.12}$$

Classification complexity is measured by the average per stage complexity,

$$\Omega(F(x)) = \frac{1}{m} \sum_{k=1}^{m} r_k(x) \Omega(f_k(x)), \tag{5.13}$$

where, using $u[\cdot]$ to denote the Heaviside step function,

$$r_k(x) = \prod_{j=1}^{k-1} u\big[F_j(x) + T_j\big], \tag{5.14}$$

is an indicator of examples that survive all stages prior to k, i.e. $r_k(x) = 1$ if $F_i(x) + T_i > 0, \forall i < k$, and $r_k(x) = 0$ otherwise. Since the average complexity is bounded by the largest weak learner complexity, this leads to a more balanced Lagrangian in (5.8) than the total complexity.

5.3.4 Cascade Boosting

The minimization of (5.8) requires the functional derivative of the Lagrangian along the direction of weak learner $g(x)$ at the current predictor $F(x)$,

$$< \delta\mathcal{L}[F], g > = < \delta\mathcal{R}_E[F], g > + \eta < \delta\mathcal{R}_C[F], g >, \tag{5.15}$$

where $< \delta\mathcal{R}_E[F], g >$ is as in (5.3). To compute the derivative of the complexity risk we note that

$$\Omega(F(x)+\varepsilon g(x)) = \begin{cases} \Omega(F(x)) & \text{if } \varepsilon = 0 \\ \Omega(F(x)+\varepsilon g(x)) & \text{otherwise.} \end{cases} \tag{5.16}$$

Defining $u(\varepsilon)$ as $u(\varepsilon) = 0$ for $\varepsilon = 0$ and $u(\varepsilon) = 1$ otherwise,

$$\begin{aligned} &\Omega(F(x)+\varepsilon g(x)) = \\ &= \Omega(F(x)) + u(\varepsilon)\left[\Omega(F(x)+\varepsilon g(x)) - \Omega(F(x))\right] \\ &= \Omega(F(x))[1-u(\varepsilon)] + u(\varepsilon)\Omega(F(x)+\varepsilon g(x)) \\ &= \Omega(F(x))[1-u(\varepsilon)] \\ &+ \frac{u(\varepsilon)}{m+1}\left[\sum_{k=1}^{m} r_k(x)\Omega(f_k(x)) + r_{m+1}(x)\Omega(\varepsilon g(x))\right] \\ &= \Omega(F(x))\left[1-u(\varepsilon)+\frac{m}{m+1}u(\varepsilon)\right] \\ &+ \frac{u(\varepsilon)}{m+1} r_{m+1}(x)\Omega(\varepsilon g(x)) \\ &= \Omega(F(x))[1-u(\varepsilon)\zeta_m] + u(\varepsilon)\frac{r_{m+1}(x)}{m+1}\Omega(\varepsilon g(x)), \end{aligned}$$

where $\zeta_m = 1 - \frac{m}{m+1}$ and we have used (5.13). Finally, since $\varepsilon g(x)$ is simply a rescaling of $g(x)$, it is assumed that $\Omega(\varepsilon g(x)) \approx \Omega(g(x))$, from which the expression above can be approximated by

$$\Omega(F(x)+\varepsilon g(x)) = \Omega(F(x))[1-u(\varepsilon)\zeta_m] + u(\varepsilon)\frac{r_{m+1}(x)}{m+1}\Omega(g(x)). \tag{5.17}$$

Furthermore, since $u(\varepsilon)$ is not differentiable, it is approximated by $\sigma(\varepsilon) \approx u(\varepsilon)$, where $\sigma(\varepsilon)$ is a differentiable function with $\sigma(0) = 0$. Under these approximations,

$$
\begin{aligned}
&< \delta \mathcal{R}_C[F], g > \\
&= \frac{1}{|S_t|} \sum_i \left[\frac{d}{d\varepsilon} \tau \Big[y_i \Omega \Big(F(x_i) + \varepsilon g(x_i) \Big) \Big] \right] \Big|_{\varepsilon=0} \\
&= \frac{1}{|S_t|} \sum_i y_i \tau' \left[y_i \Omega(F(x_i)) \right] \left[\frac{d}{d\varepsilon} \Omega \Big(F(x_i) + \varepsilon g(x_i) \Big) \right] \Big|_{\varepsilon=0} \\
&= -\frac{1}{|S_t|} \sum_i y_i \psi(y_i, x_i) \left[\frac{r_{m+1}(x_i)}{m+1} \Omega(g(x_i)) - \zeta_m \Omega(F(x_i)) \right],
\end{aligned}
\tag{5.18}
$$

where

$$
\psi(y_i, x_i) = -\tau' \left[y_i \Omega(F(x_i)) \right] \sigma'(0). \tag{5.19}
$$

Note that the derivative only depends on $\sigma'(0)$, other details of $\sigma(\varepsilon)$ make no difference. Each boosting iteration updates $F(x)$ with a step along the steepest descent direction of (5.15) within the weak learner learner pool $\mathbf{G}$,

$$
g^*(x) = \arg\max_{g \in \mathbf{G}} < -\delta \mathcal{L}[F], g > . \tag{5.20}
$$

Combining (5.3), (5.15), and (5.18) and denoting $r_i = r_{m+1}(x_i)$, $\omega_i = \omega(y_i, x_i)$, $g_i = g(x_i)$, and $\psi_i = \psi(y_i, x_i)$, this is the direction that maximizes

$$
\mathcal{D}[g] = \frac{1}{|S_t|} \sum_i y_i \left[\omega_i g_i + \eta \frac{r_i \psi_i \Omega(g_i)}{m+1} \right], \tag{5.21}
$$

where we have disregarded the term $\zeta_m \Omega(F(x_i))$ of (5.18), because it does not depend on g and plays no role in the optimization. The optimal step size for the update is

$$
\alpha^* = \arg\min_{\alpha} \mathcal{L}[F + \alpha g^*], \tag{5.22}
$$

The cascade predictor is finally updated with

$$F^{new}(x) = F(x) + \alpha^* g^*(x). \tag{5.23}$$

Note that, from (5.19), $\sigma'(0)$ is a constant that rescales all ψ_i equally. Hence, in (5.21), it can be absorbed into η. We thus assume, without loss of generality, that $\sigma'(0) = 1$. This boosting algorithm is denoted the *complexity aware cascade training* (CompACT) boosting algorithm.

5.3.5 Properties

CompACT has a number of interesting properties. First, the contribution of each training example to the complexity term in (5.21) is multiplied by r_i. Hence, only examples that survive the current cascade F contribute to the complexity term. We refer to the x_i such that $r_i = 1$ as *active* examples. Note that, given the set of active examples

$$S_a(F) = \{(x_i, y_i) \in S_t | r_i = 1\}, \tag{5.24}$$

associated with F, (5.21) can be replaced by

$$\mathcal{D}[g] = \frac{1}{|S_t|}\left(\sum_i y_i \omega_i g_i + \sum_{i|r_i=1} y_i \frac{\eta \psi_i \Omega(g_i)}{m+1}\right). \tag{5.25}$$

This complies with the intuition that examples which do not reach stage $m+1$ during the cascade operation should not affect the complexity term for that stage.

Second, most implementations of cascaded classifiers use weak learners of example-independent complexity, i.e. $\Omega(g(x_i)) = \Omega_g, \forall i$. While this does not hold for the cascade in general (different examples can be rejected at different stages), it holds for the examples in S_a, i.e. $\Omega(F(x_i)) = \Omega_F, \forall x_i \in S_a$. In this case, the complexity weights of (5.19) only depend on the label y_i. Defining $\psi^+ = -\tau'[\Omega_F]$ ($\psi^- = -\tau'[-\Omega_F]$) as the value of ψ_i for positive (negative) examples,

and π_F^+ (π_F^-) as the percentage of positive (negative) active examples, (5.21) reduces to

$$\mathcal{D}[g] = \frac{1}{|S_t|}\left(\sum_i y_i \omega_i g_i + \frac{\eta}{m+1}\Omega_g \sum_{i|r_i=1} y_i \psi_i\right) \tag{5.26}$$

$$= \frac{1}{|S_t|}\sum_i y_i \omega_i g(x_i) - \frac{\eta}{m+1}\Omega_g \frac{|S_a|}{|S_t|}\xi_F, \tag{5.27}$$

with $\xi_F = \pi_F^- \psi_F^- - \pi_F^+ \psi_F^+$. Since $|S_a|$ decreases with cascade length, the rescaling of η by $\frac{|S_a|}{|S_t|}$ gradually weakens the complexity constraint as the cascade grows. While in the early iterations there is pressure to select weak learners of reduced complexity, this pressure reduces as iterations progress. Gradually, complex weak learners are penalized less and the algorithm asymptotically reduces to a cascaded version of AdaBoost. This makes intuitive sense, since the latter cascade stages process a much smaller percentage of the examples than the earlier ones and have much less impact on the overall complexity. On the other hand, since the surviving examples are the most difficult to classify, accurate classification requires weak learner accuracy to increase with cascade length. This usually (but not always) implies that weak learner complexity increases as well because powerful features usually require heavy computation. By pushing the complexity to the later stages, the algorithm can learn cascades that are *both* accurate and computationally efficient. This effect is reinforced by the fact that $1/(m+1)$ also decreases with cascade length.

Third, for homogeneous cascades, where every weak learner g in $\mathbf{G}$ has the same complexity, the second term of (5.27) is constant and has no impact on the optimization. In this case, (5.27) reduces to (5.3) and compACT boosting reduces to AdaBoost. Since this is the setting used by most previous cascade design algorithms [161, 171, 8, 144], compACT boosting can be seen as a generalization of these procedures. More generally, the weak learner pool is frequently heterogenous but of the form $\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_M\}$, where $\mathbf{G}_m$ is a homogeneous weak learner subset and M a number of heterogeneous weak learner classes. In this case, Ω_g is constant within each $\mathbf{G}_m$, only varying across the weak learner subsets, i.e. $\Omega_g = \Omega_{\mathbf{G}_m}, \forall g \in \mathbf{G}_m$. Hence, the second term of (5.27) becomes a penalty on the weak learner subsets $\mathbf{G}_m$. While different subsets

are weighted according to their complexity, all weak learners within the same subset receive the same penalty. In this way, compACT boosting penalizes weak learner families of larger complexity but behaves like AdaBoost within each family.

5.3.6 Complexity Loss

The choice of complexity loss $\tau(v)$ determines the weight $\xi_F = \pi_F^- \psi_F^- - \pi_F^+ \psi_F^+$ in (5.27). In our experience, the details of the loss function do not have a major impact on the performance of the learned cascade. For example, because the percentage of positive training examples is usually small, the behavior of $\tau(v)$ for $v > 0$ is not critical. For negative examples, the loss should be monotonically decreasing on v, but we have not seen great differences between, say, linear and exponential decay. For simplicity, we thus adopt the loss

$$\tau(v) = \max\{0, -v\}. \tag{5.28}$$

This is similar to the SVM hinge loss [19],

$$\tau(v) = \max\{0, 1 - v\} \tag{5.29}$$

but does not enforce a penalty on positive examples of low complexity. For complexity, it matters most to distinguish between positives and negatives, penalizing positives of low complexity is not important. Instead, this loss encourages CompACT to focus on the fast rejection of negatives. Under it, $\psi_F^- = 1, \psi_F^+ = 0$ and $\xi_F = \pi_F^-$. Note that this makes the weight $\frac{|S_a|}{|S_t|}\xi_F$ of (5.27) equal to the ratio between the negative examples that are active and the total number of examples. This decreases as the cascade grows.

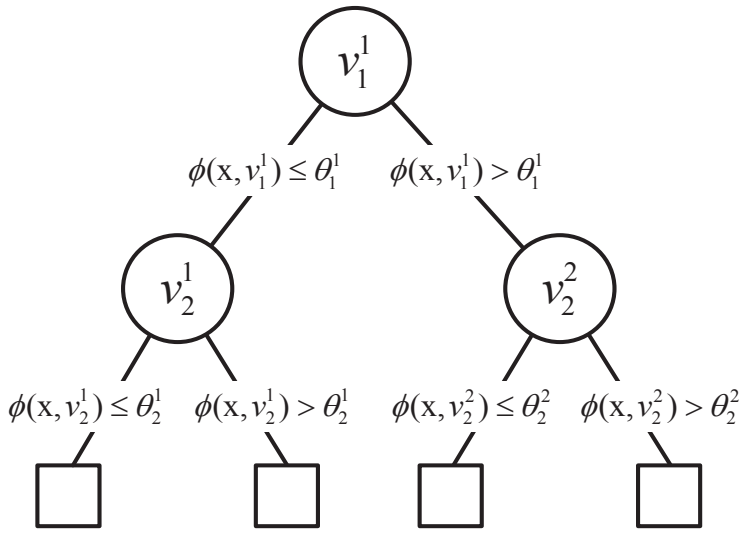


Figure 5.1: Decision tree of depth 2. Circles represent classifier nodes, squares terminal nodes. $\phi(x, v)$ is the feature of x used at node v, θ a threshold.

5.3.7 Weak Learners

CompACT supports the design of cascades with many types of weak learners. In this work, we follow the predominant trend in the literature and use binary decision trees.

Decision trees: As shown in Figure 5.1, binary decision trees have two node types: classifier and terminal. Node v implements a classifier $g(x, v)$ by computing a feature $\phi(x, v)$ of sample x and comparing it to a threshold $\theta(v)$. The sample is then forwarded to one of the two children of the node, according to the results of this comparison. In this way, each sample follows a path from the root to a terminal node, transversing a single classifier node at each depth level. No classification or feature computation is performed in terminal nodes. Hence, in what follows, tree depth refers to classifier nodes only.

A binary decision tree g of depth L with classifier nodes $\{v_1^1, \cdots, v_l^i, \cdots, v_L^{n_L}\}$, where $n_l = 2^{l-1}$ is the number of nodes at depth l, has expected computation cost

$$\Omega(g) = \sum_{l=1}^{L} \sum_{i=1}^{n_l} p_l^i c_l^i, \tag{5.30}$$

Algorithm 1 Complexity-Restricted Tree Learning

Input:
Feature pool $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$; training samples $S_t = \{(x_i, y_i)\}$.
Output:
The optimal decision tree g^*.
for $k = 1 : M$ **do**
  Build the optimal homogenous decision tree g_k, using features from ϕ_k only and (5.4).
end for
Select the decision tree g^* from $\{g_k\}$ that maximizes (5.27).

where $p_l^i = p(v_l^i|x)$ is the probability that node v_l^i is traversed by sample x and c_l^i is the feature computation cost of node v_l^i. The probabilities p_l^i have some properties that derive from the tree structure, e.g. 1) if v_l^k is the parent of v_{l+1}^j then $p(v_{l+1}^j|x) = p(v_{l+1}^j|x, v_l^k)p(v_l^k|x)$; 2) $\sum_{i=1}^{n_l} p_l^i = 1$ for any depth l. Property 1) implies that $p(v_{l+1}^j|x) \leq p(v_l^k|x)$, i.e. the probabilities decrease with depth, and property 2) guarantees that the root node is traversed with probability $p(v_1^1|x) = 1$ for all x.

In this work, the feature pool ϕ used to build decision trees is composed of M classes, $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$, where all features of class k have computational cost C_k. It follows that $c_l^i \in \{C_k\}_{k=1}^M$. Two strategies for learning decision trees, which are discussed next, were considered.

Complexity-restricted Tree: Under the first strategy, all decision trees use homogeneous features. At each boosting iteration, a decision tree g_k is built from feature sub-pool ϕ_k. The learning algorithm then chooses the best of the M trees g_k. Under this strategy, (5.30) reduces to

$$\Omega(g_k) = \sum_{l=1}^{L} \sum_{i=1}^{n_l} p_l^i C_k = LC_k, \tag{5.31}$$

where we have used property 2) above. Hence, given k, $\Omega(g_k)$ is a constant and the optimal weak learner of (5.20) is identical to that of (5.4), i.e. that chosen by AdaBoost. Hence, the learning of each cascade stage can be divided into two steps: 1) learn M homogeneous decision trees, by using the AdaBoost rule of (5.4) to find the optimal tree g_k for each each feature pool ϕ_k, and 2) select the homogeneous tree $g^* \in \{g_k\}$ that maximizes the complexity aware objective of (5.27).

Algorithm 2 Complexity-Sensitive Tree Learning

Input:
Feature pool $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$; training samples $S_t = \{(x_i, y_i)\}$; and tree depth L.
Output:
The optimal decision tree $g^* = \{g_1^1, \cdots, g_L^{n_L}\}$.
for each classifier node v **do**
1. Collect the training samples S_v that fall into node v, and compute the probability p_v that node v is traversed by a sample.
2. Find the node classifier g_v, by selecting the combination of feature from ϕ and threshold $\theta(v)$ that maximizes (5.32).
end for

The procedure is summarized in Algorithm 1.

Complexity-sensitive Tree: The second strategy optimizes the trade-off between complexity and accuracy even within each decision tree. Since global tree optimization is difficult, we use the popular local recursive optimization, where nodes are optimized sequentially. Consider (5.27) and the active sample $|S_a|$ arriving at current stage. Using S_v to denote the samples that reach node v, the empirical probabilities p_v are estimated with $p_v = \frac{|S_v|}{|S_a|}$. The binary classifier $g(x, v)$ implemented at node v is then learned with the following variant of (5.27)

$$\mathcal{D}[g_v] = \frac{1}{|S_v|} \sum_{i \in S_v} y_i \omega_i g_v(x_i) - \frac{\eta}{m+1} \frac{|S_a|}{|S_t|} p_v \xi_F \Omega_{g_v}, \tag{5.32}$$

where we use g_v to denote $g(x, v)$. The optimization procedure is described in Algorithm 2. Note that g_v can choose a feature from any of the feature pools ϕ_k. Hence, the tree is usually heterogenous and has two main properties. First, because the complexity penalty decreases with node probability, less visited nodes tend to use more complex features. Second, from property 1) above, deeper nodes have smaller probabilities than earlier ones, and feature complexity increases towards the tree leaves. Hence, early and popular nodes tend to use low complexity features, while features of higher complexity tend to be chosen for late or less popular nodes.

Algorithm 3 Embedded Bootstrapping

Input:
Training images set I; numbers $\{N_1, \cdots, N_K\}$ of weak learners at different stages, where $M = \sum_{k=1}^{K} N_k$.
Initialization:
Empty detector $\mathcal{F} = \{\}$; initial training set S_0.
Output:
Final detector $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_K\}$.
for $k = 1 : K$ **do**
1. Use S_{k-1} to learn a detector $\mathcal{F}_k$ of N_k weak learners.
2. Embed $\mathcal{F}_k$ in detector $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$.
3. Run $\mathcal{F}$ to collect a new hard training set S_k.

end for

5.3.8 Bootstrapping and Thresholds

Bootstrapping is critical for effective object detection [38, 29]. A popular strategy is to collect hard negatives for an immature detector and use them to train a less immature detector. This is repeated for several rounds, until the detector is mature enough [29, 127, 192, 121, 5]. Usually, performance saturates after 4 to 5 bootstrapping rounds. A problem of this strategy is that the final training samples are not representative of the sample distribution of natural images. In fact, in the final bootstrapping round, most negatives are similar to the positives. Hence, there is no guarantee that the final detector will correctly classify the samples rejected by previous immature detectors.

To circumvent this, we use the bootstrapping strategy of Algorithm 3, which is similar to the methods of [42, 153]. The final detector is not the one learned in the final bootstrapping round. Instead, the detector learned in current round is embedded into the detector learned from all previous rounds, using (5.12). In our experiments, this strategy is more resistant to over-fitting than the classical one, sometimes outperforming it by a large margin.

Another important factor for cascade performance is the threshold used to reject negative examples. While thresholds can be learned [8], this increases the complexity of learning the complexity-aware cascade stages. In this work, the thresholds are selected as in [29], using a simplified version of the soft cascade of [8]. This consists of setting, in (5.14), $T_1 = 1$ and

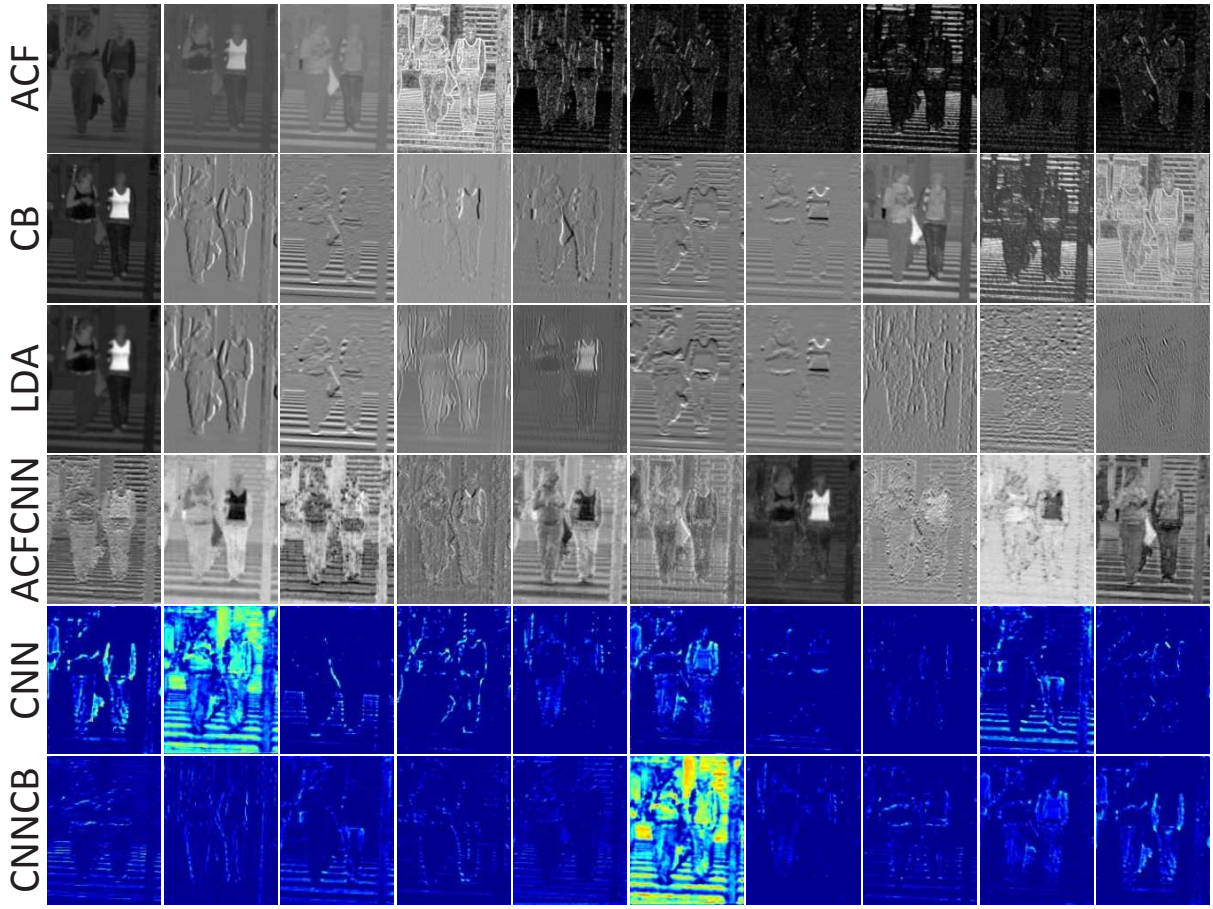


Figure 5.2: Sample of the feature channels generated on a pedestrian image.

$T_{j+1} - T_j = \Delta, \forall j > 1$, where Δ is usually a small number, e.g. 0.005.

5.4 Pedestrian Detector Design

In this section, we discuss various details of the proposed pedestrian detector.

5.4.1 Heterogeneous Features

CompACT boosting seeks the optimal trade-off between accuracy and complexity, at each cascade stage. This is most effective when the feature pool contains features of various complexities. Our implementation draws such features from two main sources, the handcrafted

aggregate feature channels (ACF) of [29] and a CNN feature set. Different filter sizes, computational mechanisms, platforms, etc. contribute to the diversity of feature complexities. A sample of the responses of some of these features to a pedestrian image is shown in Figure 5.2. Note the diversity of details that the features highlight. This is unlike most previous works in the cascade literature, where most detectors use a single feature family.

In the literature, it is common to pre-compute a large number of feature responses at all image locations, before detection [121, 192, 127]. This, however, has unfeasible complexity for large feature pools (e.g. the 200,000∼500,000 features extracted per patch in [192, 127]) or when features are computationally intensive (e.g. the CNN features of [86, 150]). In these cases, it is neither tractable nor necessary to pre-compute all features at all locations. For example, a cascade of 2048 decision trees of depth 2, evaluates at most 4096 features per patch. Since the cascade rejects most candidate patches after a few stages, the most intensive features (e.g. CNN) are unlikely to be needed at most image locations. Hence, while pre-computation is useful for low-complexity features, complex features should be evaluated as necessary. We refer to the former as *pre-computed* features and the latter as computed *just-in-time* (JIT).

Pre-computed Features: Our pre-computed feature set consists of ACF [29], mostly due to its computational efficiency. Following [29], we extract 10 LUV+HOG channels. In total, there are $16\times8\times10$=1,280 ACF features per pedestrian patch.

Just-in-time Features: The JIT pool contains several feature subsets as introduced below.

SS: The self-similarity (SS) features of [148] capture the difference between local patches and have achieved good performance on edge detection tasks [94, 33]. Following [94, 33], we compute SS features on a 12×6 grid of the 16×8 ACF patch. This results in $\binom{72}{2}\times10=25,560$ SS features per pedestrian patch.

CB: Checkerboard features (CB) are obtained by convolving the ACF channels with a set of checkerboard filters. [192] has shown that a simple set of such features could achieve very

Figure 5.3: Eight 2×2 checkerboard-like filters used in this work. Red (Green) is used to represent value +1 (-1).

good performance for pedestrian detection. Based on their observation that the number of features determines performance (rather than feature type), we adopt the set of 8 simple 2×2 checkerboard filters of Figure 5.3. In total, there are 16×8×80=10,240 CB features per pedestrian patch.

LDA: Locally decorrelated HOG features, computed with linear discriminant analysis (LDA), have shown some superiority for object detection over HOG features [56]. [121] showed that the computation of these features on ACF channels leads to a big improvement over ACF. We adopt this feature family, with filter size of 3×3. In total, there are 16×8×40=5,120 LDA features per pedestrian patch.

ACFCNN: SS, CB and LDA features are responses of simple hand-crafted filters to ACF features. A set of filters of higher complexity is also learned with a shallow CNN. This has the ACF feature channels as input and consists of a convolutional and a fully connected (fc) layer. Since ACF channels vary quite a bit in magnitude, they are passed through a batch normalization [73] layer before being fed to the CNN. The convolutional layer contains 32 filters of size 1×3×10 and 3×1×10, where 10 is the number of ACF channels. These filters have the same complexity and are denoted as ACFCNN. Although GPUs are used to train ACFCNN parameters, they are relatively simple filters and we use a CPU for their computation. In total, there are 16×8×64=8,192 ACFCNN features per pedestrian patch.

Small CNN: Beyond operators defined over ACF channels, we consider a set of CNN features extracted directly from the image to classify. The CNN has eight convolutional layers and one fc layer, whose details are given in Figure 5.4. It was originally trained on 64×64 ImageNet images [143] and then fine tuned on a pedestrian dataset of 64×32 images. For feature extraction,

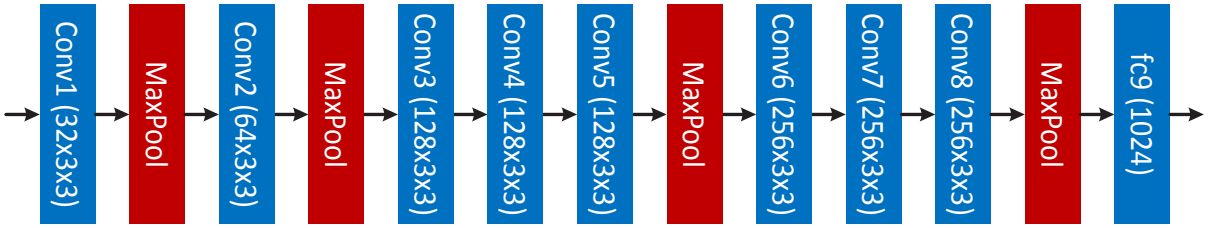


Figure 5.4: CNN architecture used to extract small CNN features.

we use the output of the 5^{th} and 7^{th} convolutional layers. Similarly to ACF, these can be seen as CNN feature channels and are denoted as CONV5 and CONV7, respectively. Inspired by the good performance and simplicity of the checkerboard features on ACF, we also compute them on the CONV5 feature channels. These are denoted CONV5CB features. Since CONV5 and CONV7 features are strong and can drive boosting towards over-fitting (especially CONV7), we only retain the $1/2$ channels of CONV5 and the $1/4$ channels of CONV7 that are more frequently used during learning. After the cascade is trained, we prune the unfrequently used channels and retrain it. Overall, there are $16\times8\times64$=8,192 CONV5 features, $8\times4\times64$=2,048 CONV7 features and $16\times8\times512$=65,536 CONV5CB features per pedestrian patch.

5.4.2 Feature Complexity

Different facets of feature complexity, such as number of computations, computation cost, energy consumption, or speed, are of importance depending on the application. For features computed on CPUs, these variables are linearly related, and any of them can be used as a general measure of complexity. A reasonable choice is the number of floating-point operations (FLOPs) required by feature computation. However, complexity can depend on the implementation. For example, a Haar feature has complexity linear in the size of the image region it covers, but this leads to substantial redundant computation when overlapping features are used. The problem can be avoided by using the integral image of [161], which enables constant complexity (four integral image operations) for all features. This type of problem is much more complex for

Table 5.1: Feature complexity and processing unit

	ACF	SS	CB	LDA	ACFCNN	CONV5	CONV7	CONV5CB
FLOPs	1	2	4	9	30	5.84×10^7	8.67×10^7	5.84×10^7+4
Unit	CPU	CPU	CPU	CPU	CPU	GPU	GPU	GPU+CPU

CNN features computed on GPUs. In this case, speed is optimized when features are computed in parallel. It is frequently faster to compute convolutions over the entire image, even if this computation is not needed in some areas, than to adapt the computation to the detection needs. This is, however, very inefficient in terms of energy consumption. Hence, for GPU-based CNN features, the two facets of complexity are not linearly related. The problem could be solved by shutting down individual CNN units, depending on the spatial location of their receptive field, but this is usually not supported by deep learning libraries. Hence, it is difficult to define a universal measure of complexity, of interest for all applications. This is even more complex for mixed CPU/GPU implementations, since the cost and energy consumed per FLOP are very different for the CPU and GPU architectures. In this case, the complexity measure can depend on the particular hardware implementation of the detector. We avoid this problem by introducing a generic but approximate measure of computational complexity, which relies on FLOPs to measure the complexity of CPU features and introduces the notion of "trigger complexity" for those computed on GPUs. The FLOPS and processing unit required by the various features used in this work are shown in Table 5.1.

ACF-based Feature Complexity: ACF features underlie SS, CB, LDA and ACFCNN, and are pre-computed before detection. The pre-computation cost, which occurs before cascade evaluation, can be ignored reducing complexity to a memory access. We assign to this a complexity of 1. SS, CB LDA and ACFCNN features are JIT features. Beyond ACF, they require a number of FLOPs proportional to their filter sizes. As shown in Table 5.1, their complexities are 2, 4, 9 and 30 respectively.

Trigger Complexity: The complexities of CNN and hand-crafted features are of a

different nature. While CNN features are computed JIT, current libraries makes it inefficient to compute features individually. If CNN features are needed to classify an image region, it is significantly more efficient to evaluate all layers over the entire region than only the responses of the layers needed per sub-region. This makes it difficult to assign a complexity per feature. Instead, we rely on the concept of a trigger complexity Ω_{cnn}.

When (5.27) is evaluated for a previously unused CNN feature, $\Omega_g = \Omega_{cnn}$. Similarly to ACF, once the feature is computed its complexity is set to $\Omega_g = 1$. Different CNN features have different trigger complexities, which reflects their requirements in GPU FLOPS. In our implementation, CONV5 and CONV7 have trigger complexities of $\{\alpha_5\Omega_{cnn}, \alpha_7\Omega_{cnn}\}$, where $\alpha_5 : \alpha_7 = 1 : 1.48$. This is summarized in Table 5.1. When CNN architectures are independent, the same holds for their trigger complexities. However, CNN features are frequently dependent, e.g. CONV5 and CONV7, which are features from the same network, with CONV7 requiring the computation of CONV5. To account for this, once CONV5 is computed trigger complexities become $\{0, (\alpha_7 - \alpha_5)\Omega_{cnn}\}$ and once CONV7 is computed they become $\{0, 0\}$. Once these features are triggered, complexity becomes $\{1, 1\}$ for both CONV5 and CONV7. CONV5CB features, obtained by checkerboard filtering the responses of CONV5, have trigger complexity equal to CONV5. Their after-trigger complexity is 4, the number of FLOPs required by checkerboard filters.

5.4.3 Embedding Large CNN Models

Large CNNs [86, 150, 61] are popular in computer vision. While, theoretically, they could be used in any cascade stage, this makes the iterative boosting optimization too computationally intensive. It is practical, however, to use a large CNN as the *final* cascade stage. This can, in principle, be done for any CNN detector in the literature. We next discuss the embedding of a few popular methods.

R-CNN: The R-CNN detector[51] applies a CNN to object proposals produced by an

independent algorithm. It is trained with a sequential process of CNN fine-tuning, feature saving, SVM classifier training and bounding box regression training. This requires substantial human engineering, time and memory, due to the discrepancy between CNN classification and SVM detection scores. Some components, e.g. feature saving and SVM training, can be avoided by using CNN scores as detection scores. Due to the low computational efficiency of the R-CNN [51], embedding it in a CompACT cascade, which is already very discriminant and produces few detection candidates, can be very beneficial.

Fast-RCNN: The Fast-RCNN addresses many limitations of the R-CNN by sharing computation across proposals at different locations and scales. It requires a single forward image pass to produce all detections. This makes it much more efficient than the R-CNN. The original Fast-RCNN [50] was trained with third-party generic object proposals [158], which is not efficient nor effective for pedestrian detection. Better results could, in principle, be obtained by training a Faster-RCNN model [140] to detect pedestrians first, and then replace the RPN proposals by CompACT cascade proposals. The latter can, in this case, be seen as a strong proposal mechanism for the Fast-RCNN. To test this premise, we used Fast-RCNN detectors based on AlexNet, VGG-Net and ResNet-101.

MS-CNN Despite the good Faster-RCNN results for general object detection, its pedestrian detection performance can be weak. On the contrary, the MS-CNN introduced in Chapter 2 is much stronger on pedestrian detection. To use the MS-CNN with CompACT, the proposal generation of Figure 2.3 is ignored. Only the detection network of Figure 2.4 is embedded in the CompACT cascade. This is similar to the embedding of the Fast-RCCN (of VGG-Net), but uses a more accurate pedestrian detector. It could thus enable better trade-offs between accuracy and complexity.

Embedding Mechanisms: The simplest way to embed a large CNN in a CompACT cascade is the "Proposal" mechanism of the R-CNN [51]. The CompACT cascade generates proposals, which are fed to the CNN. The final detection scores are produced by the CNN alone.

This strategy assumes that the proposal detector is much less discriminant than the large CNN. This is frequently not true for CompACT cascades which, at least for pedestrian detection, can even be stronger than the large CNN. To address this, we propose an "Embedded" mechanism, where the large CNN acts as the final weak learner g of (5.23). Note that this has no loss of optimality, if α is learned with (5.22). While avoiding the computationally intensive fine-tuning of the CNN to the cascade proposals, it guarantees that the cascade is still *optimal* in an end-to-end manner. The CNN is simply a descent direction of (5.20) that is unavailable to prior stages. In summary, "Embedding" differs from the "Proposal" mechanism in that 1) not only the bounding boxes but also the confidence scores of the cascade are forwarded to the deep CNN, and 2) the combination of the proposal mechanism (cascade) and deep CNN is optimal under the well defined risk of (5.8).

It should be noted that the stride (step between window evaluations) of CompACT is usually small, e.g. 4 pixels, while that of large CNN feature maps is relatively large and dependent on the network architecture, e.g. 16 pixels for the CONV5 features of VGG-Net. However, this is not a problem when the R-CNN or Fast-RCNN are embedded in the CompACT cascade, because the networks operate on object proposals. It suffices to feed the proposals to the network by image warping (R-CNN) or RoI pooling (Fast-RCNN). Another potential problem is that, despite its small stride, the cascade is not always strong enough to guarantee good localization, sometimes scoring the ground-truth window lower than neighboring windows. This, however, is not a problem for Fast-RCNN detectors, which can recover from poor proposal localization by relying on bounding box regression [50, 140]. Since the latter has a trivial computational cost, it is always used when either the Fast-RCNN or MS-CNN are embedded in a CompACT cascade.

It remains to decide what CompACT proposals are forwarded to the CNN. As is common in the literature, when the cascade is used by itself, its output is post-processed with non-maximum suppression (NMS). Since there are frequently many overlapping proposals, this drastically reduces the number of deep CNN evaluations, the costliest stage from a computational

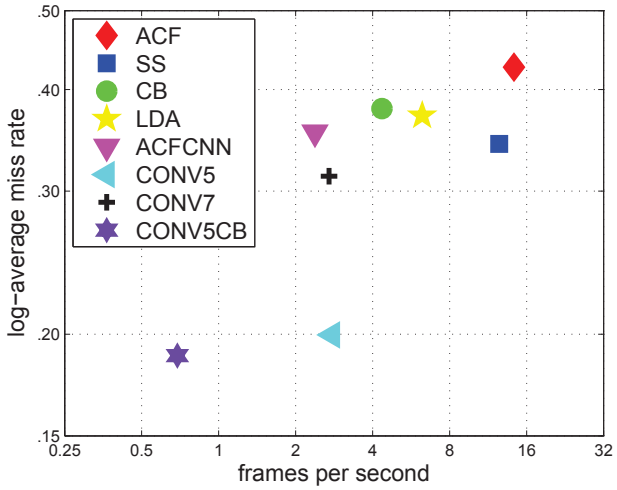
point of view. On the other hand, it is sub-optimal to insert the NMS operation between the two stages from a detection point of view. We study the impact of embeddings both with and without NMS on the detection accuracy/complexity in the experimental section.

5.5 Experiments

In this section, we discuss various experiments performed to evaluate CompACT pedestrian detection cascades, using the Caltech [32] and KITTI [43] datasets.

5.5.1 Experimental Setting

We started by performing various ablation studies on Caltech, all using cascades of 2048 decision trees of depth of 2. These were learned with 7 bootstrapping rounds, using $\{32, 96, 128, 256, 512, 512, 512\}$ decision trees. The training and testing sets were as in [29]. A set of final experiments was then conducted using cascades of 4096 decision trees of depth of 5 and 9 bootstrapping rounds ($\{32, 96, 128, 256, 512, 768, 768, 768, 768\}$ trees). These were performed on both Caltech and KITTI (a decision tree depth of 4 was used in KITTI, due to its much smaller number of pedestrian instances). The pedestrian template size was 64×32 including a contextual border region of 14×12, as in [29]. On Caltech, we used the training set of [121], on KITTI we used the standard training set. The standard evaluation metrics were used on each dataset: MR (log-average miss-rate) on Caltech, and mAP (mean average precision) on KITTI. Since the minimum pedestrian height on KITTI is half of that of Caltech (25 pixels), the original test images were upsampled by 2. The detected bounding boxes (minimum height of 50) were then downsampled by 2. All experiments were run on a single CPU core (2.10GHz) of an Intel Xeon E5-2620 server with 64GB of RAM. An NVIDIA Tesla K40M GPU was used for CNN computations, with the remaining computations performed on CPU.



feature	MR	time (s)
ACF-based		
ACF	42.6	0.07
SS	34.29	0.08
CB	37.89	0.23
LDA	37.15	0.16
ACFCNN	35.41	0.42
CNN-based		
CONV5	19.96	0.36
CONV7	32.34	0.37
CONV5CB	18.83	1.35

Figure 5.5: Cascade accuracy v.s. complexity for different features.

5.5.2 Homogeneous Cascade Comparison

We started with homogeneous cascades of a single feature family, the predominant architecture in the literature. In this case, the complexity penalty of (5.27) is equal for all weak learners, and CompACT reduces to AdaBoost. Figure 5.5 compares homogeneous cascades based on ACF, SS, CB, LDA, ACFCNN, CONV5, CONV7 and CONV5CB. Note the improved performance of our ACF cascade implementation over [29], due to the bootstrapping strategy of Section 5.3.8.

Clearly, SS outperforms the other ACF-based features (ACF, CB, LDA, ACFCNN), achieving higher accuracy *and* speed. CB and LDA are more discriminant than ACF, but more complex. ACFCNN is slightly more accurate than CB, but its heavy features produce the slowest ACF cascade. The cost/benefits of CNN features are shown on the lower table. CONV5 has higher accuracy than all ACF-based features but five times the complexity. CONV7 has much worse detection performance than CONV5. We believe this is due to a coarser sliding window stride, since (see Figure 5.4) CONV7 is downsampled twice more than CONV5. CONV5CB has the best detection, but only a marginal gain over CONV5 and four times the computation. Figure 5.5 also shows that the spectrum of feature accuracies v.s. complexities is quite broad.

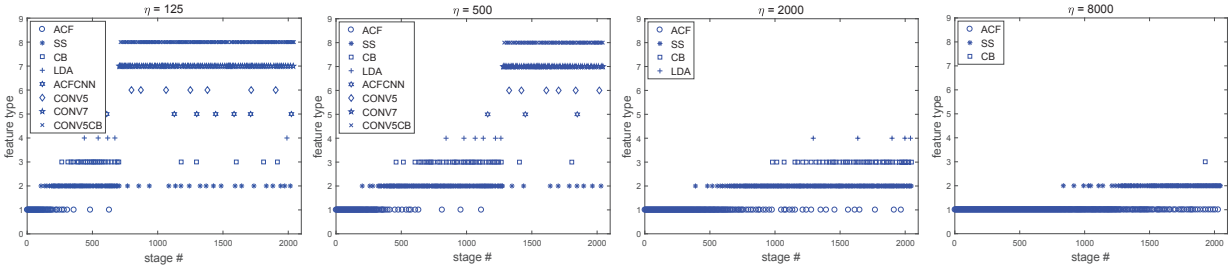


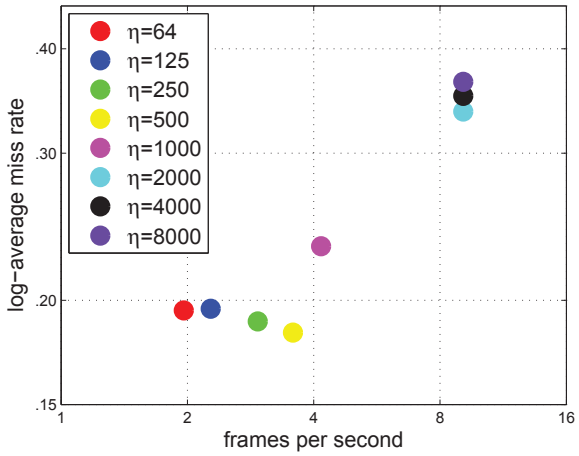
Figure 5.6: Stage configuration learned with different trade-off coefficient η. Only one in five stages is shown.

This enables CompACT to select features of optimal accuracy vs complexity trade-off at each cascade stage.

5.5.3 CompACT Cascade Configuration

To test the ability of compACT to trade-off accuracy for complexity, we started by considering cascades of complexity-restricted trees (Section 5.3.7). Figure 5.6 illustrates the configuration of cascades learned with different η in (5.8), showing the feature type selected every fifth stage. Consider the case of $\eta = 500$. The cheapest features (ACF) were the only selected for the first 200 stages, and rarely used after stage 500. This suggests that ACF features are very efficient but not very discriminant. A better trade-off between the two goals is achieved by SS features, which were selected throughout the training process. Note that SS features are competitive even for the later cascade stages, suggesting that they can be very discriminant in spite of their simplicity. Similarly, CB features were selected across a large range of cascade stages. This is unlike LDA and ACFCNN, whose rare selection suggests a weak discrimination vs. complexity trade-off. More surprisingly, CONV5 features were also rarely selected, with CONV7 and CONV5CB dominating the late stages. This suggests that CONV7 and CONV5CB are more discriminant. Recall that, while CONV5 is more efficient in Table 5.1, CompACT boosting weighs complexity less heavily than discrimination in the late stages.

Figure 5.6 also shows that the optimal cascade configuration depends strongly on the



η	MR	time (s)
64	19.45	0.51
125	19.54	0.44
250	18.87	0.34
500	**18.29**	0.28
1000	23.21	0.24
2000	33.65	0.11
4000	35.13	0.11
8000	36.52	0.11

Figure 5.7: Accuracy v.s. complexity for different trade-off coefficients η.

Lagrange multiplier η that controls the trade-off between accuracy and complexity. Recall, from (5.7)-(5.8), that η has an inverse relationship with the complexity bound γ. Hence, a smaller η (larger γ) should produce a slower and more accurate cascade, and vice versa. This is supported by Figure 5.6, where large η leads to a cascade composed exclusively of low complexity features, while a small η allows the use of more complex features and the use of those features earlier on in the cascade. The overall trade-off between accuracy and complexity is shown in Figure 5.7, which compares cascades learned with different η. Small η, which as shown in Figure 5.6 induce CompACT to select expensive CNN features in the early stages, produce slower but more accurate cascades. Larger η, which prevent the selection of expensive features, produce faster but less accurate detectors. In the limit of $\eta \rightarrow \infty$ only the cheapest features are chosen throughout the cascade, and CompACT reduces to AdaBoost on the ACF feature pool. This is the method of [29].

Figure 5.7 shows that between the two extremes of η, cascade speeds vary from about 2 to about 10 fps. Interestingly, accuracy plateaus as speed decreases. Note that accuracy does not improve as η decreases beyond $\eta = 500$, even though speed continues to decrease. Since the knee of the curve corresponds to the lightest cascade with the best accuracy on this dataset, this is the cascade of optimal trade-off between accuracy and complexity. The fact that cascades

with heavier features have weaker detection rate suggests some over-fitting and, as discussed in Section 5.2, that the complexity penalty acts as a regularizer. In these experiments the optimal trade-off was achieved with $\eta = 500$.

One sensible question is whether CNN features would suffice to implement all cascades. After all, ACF features perform convolutional operations and a well trained CNN model should be able to learn them. Figure 5.6 suggests this is not the case. This is likely because ACF features were designed with the explicit purpose of computational efficiency. Although they cannot guarantee high accuracy, they achieve a strong trade-off between accuracy and run-time. Note that, in Figure 5.6, they are always chosen to implement the early cascade stages. On the other hand, CNN features are never chosen for cascades that emphasize computational efficiency (large η). While the CNN could, in principle, learn the ACF features, this would require 1) a CNN learning algorithm that somehow penalized feature complexity, which we are not aware of, and 2) probably some form of hand-coding, through specification of the architecture of early CNN layers.

5.5.4 Benefits of optimal accuracy/complexity trade-off

We next compared CompACT to algorithms for learning cascades of heterogenous features. Since there is no literature in this area, we considered two baselines. The first learns a cascade without complexity constraints ($\eta = 0$ in (5.27)), and is denoted as "Boosting". It is equivalent to applying existing cascade learning algorithms to the diverse feature set considered in this work[1]. The second attempts to "hand-code" the behavior of CompACT, restricting the unconstrained boosting algorithm ($\eta = 0$) to use certain feature types in different cascade stages. This restriction is based on feature complexity: features were ranked by complexity and used sequentially. ACF features were used in the first 512 stages and the other feature types used equally in the remaining

[1] Since, in this case, all feature channels must have the same size (downsampling rate), CONV7 features were not used in this section.

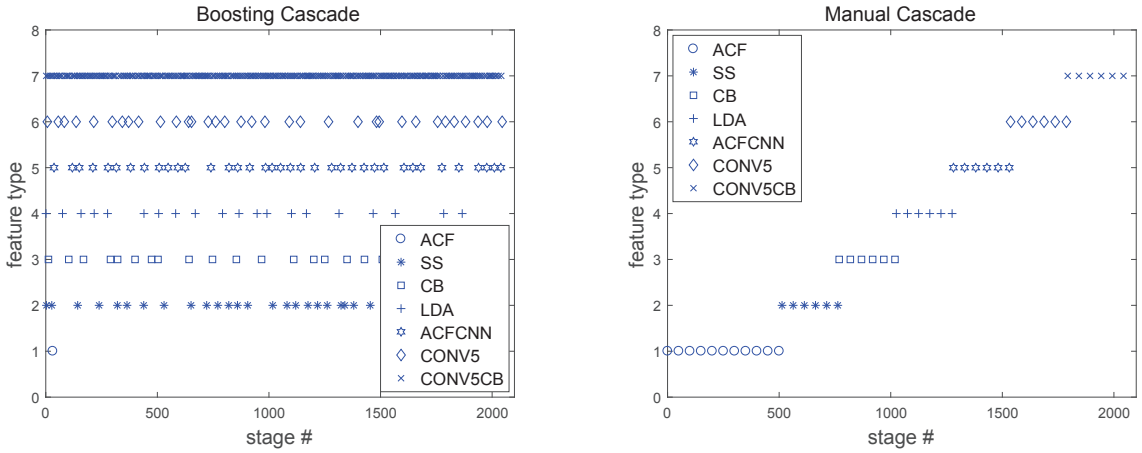


Figure 5.8: Stage configuration for "Boosting" and "Manual" cascades.

stages. The resulting cascade is denoted as "Manual".

Figure 5.8 shows the stage configurations of the "Boosting" and "Manual" cascades. Since "Boosting" does not penalize complexity, it frequently selects CONV5 and CONV5CB features for the early cascade stages. Note that this makes it too costly to compute features JIT, as discussed in Section 5.4.1. Instead, all channel features are computed at the beginning, enabling feature sharing. Only SS is computed JIT. Despite this feature sharing, the "Boosting" cascade was still several times slower than the CompACT cascade. This can be seen in Table 5.2, which compares CompACT ($\eta = 500$), "Boosting", and "Manual". The two sides of the table differ in that only ACF-based features were used on the left, while both these and the small CNN based features were used on the right. In both cases, the "Manual" cascade has low complexity but poor accuracy. "Boosting" learns a more accurate but significantly more complex cascade. CompACT has the best trade-off between accuracy and complexity. Note also that the introduction of the small CNN enabled substantially better cascades.

It is also instructive to compare these results to those of the homogeneous feature cascades commonly used in the literature, shown in Figure 5.5. When compared to the heterogeneous CompACT cascades, all homogeneous cascades performed poorly. The CompACT cascade of multiple ACF features had higher accuracy than all homogeneous ACF-based feature cascades and

Table 5.2: Comparison to "Boosting" and "Manual" cascades.

Method	ACF-based			ACF-based+Small CNN		
	Boosting	Manual	CompACT	Boosting	Manual	CompACT
MR	32.09	34.39	33.25	17.39	23.51	18.29
time (s)	0.76	0.11	0.11	2.46	0.23	0.28

Table 5.3: Comparison of complexity restricted and sensitive trees.

η		125	250	500	1000	2000	4000	8000	16000
restricted	MR	30.97	29.23	29.56	29.84	30.35	30.94	30.59	31.68
	Time (s)	0.105	0.104	0.103	0.102	0.101	0.101	0.100	0.100
sensitive	MR	29.32	28.30	29.33	29.78	30.16	30.28	30.91	31.31
	Time (s)	0.109	0.111	0.103	0.104	0.104	0.101	0.102	0.105

was faster than most. The CompACT cascade of multiple ACF+Small CNN features had the best overall detection performance. Not only its detection accuracy beat that of the best single-feature cascade (CONV5CB in Figure 5.5), it was also 5 times *faster*. These results illustrate the benefits of combining features of multiple complexities and optimizing the trade-off between accuracy and complexity.

5.5.5 Complexity Restricted vs. Sensitive Trees

So far, we have reported on heterogenous feature cascades using complexity restricted trees as weak learners. The next set of experiments compared these to cascades of complexity sensitive trees (Section 5.3.7). Here, we found that the inclusion of CNN features could lead to some instability. Although these features have high trigger complexity, they can have small penalty under (5.32) if the corresponding nodes are visited with small enough probability. Since small probabilities are difficult to estimate, this can lead to significant errors in node complexity estimates. In result, CNN features can be selected too often in early cascade stages. To avoid this, we used only ACF based features in these experiments. Decision trees of depth 3 were also used to magnify the differences between the two approaches.

As shown in Table 5.3, for equal η, complexity-sensitive trees are more accurate but

Table 5.4: Faster-RCNN baseline results.

Method	Alex	VGG	ResNet	MS-CNN
MR	30.29	15.59	14.91	10.0

slower than complexity-restricted trees. However, the differences are quite small, indicating there is no need to enforce trade-off optimization inside decision trees. This is a positive finding, in three ways. First, it shows that CompACT is quite robust, making it unnecessary to carefully optimize each weak learner. Second, it is in agreement with the boosting philosophy, whose point is exactly to aggregate "weak" (i.e. sub-optimal) learners . The requirement for a very detailed optimization of the decision tree would violate this principle. Third, complexity-restricted trees are simpler and easier to implement than complexity-sensitive trees, making the approach more practical. Given all this, we use complexity-restricted trees in what follows.

5.5.6 Embedding Large CNN models

We next considered the benefits of embedding large CNNs in CompACT cascades. A cascade of 4096 decision trees of depth 5 was used in these experiments[2]. Table 5.5 compares the performance of the CompACT cascade with small CNNs (denoted CompACT), versus its combination with the R-CNN and Fast-RCNN embedding of several popular models, including AlexNet [86], VGG-Net [150], and ResNet-101 [61], and the recently proposed SDS-RCNN [9] (excluding its RPN part). As discussed in Section 5.4.3, the large CNNs operated only on proposals generated by CompACT. The Fast-RCNN models were produced by removing the proposal generator of the Faster-RCNN models, whose baseline results are shown in Table 5.4. We investigated the benefits of 1) applying NMS to the cascade output before vs. after application of the large CNN, and 2) simply passing cascade detections to the CNN as a set of proposals (denoted "Proposal") vs. actually embedding the large CNN in the cascade with (5.12), to obtain

[2]Because the combination of CONV7 features and these deeper trees can sometimes over-fit, CONV7 features were not used in this section.

Table 5.5: Embedding of large CNNs in CompACT cascades. "+" denotes additional time, after embedding the CNN, over CompACT.

CompACT	
MR	15.27
time (s)	0.34

R-CNN								
	after NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	SDS	Alex	VGG	ResNet	SDS
MR	17.33	12.05	13.23	14.66	14.80	11.37	12.31	11.82
time (s)	+0.02	+0.06	+0.07	+0.07	+0.02	+0.06	+0.07	+0.07
	before NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	SDS	Alex	VGG	ResNet	SDS
MR	17.09	10.61	12.40	11.38	13.46	9.61	10.65	9.44
time (s)	+0.13	+0.21	+0.72	+0.54	+0.13	+0.21	+0.72	+0.54

Fast-RCNN								
	after NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	MS-CNN	Alex	VGG	ResNet	MS-CNN
MR	27.55	14.83	15.93	12.65	20.69	11.95	13.00	10.91
time (s)	+0.03	+0.24	+0.26	+0.24	+0.03	+0.24	+0.26	+0.24
	before NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	MS-CNN	Alex	VGG	ResNet	MS-CNN
MR	25.32	13.23	12.64	10.12	18.49	10.01	10.65	9.06
time (s)	+0.03	+0.24	+0.28	+0.24	+0.03	+0.24	+0.28	+0.24

an integrated detector (denoted "Embedded").

A number of interesting observations follow from the table. First, the theoretically sounder embedding of "Embedded" outperformed the more ad-hoc "Proposal" mechanism, in all cases. This shows that CompACT cascade scores and large CNN scores contain complimentary information. Second, CompACT is already a good pedestrian detector. Under "Proposal," Alex does not improve on the CompACT cascade. In fact, its MR (15.27) is comparable to the Faster-RCNN result of ResNet (14.91) in Table 5.4. Third, higher detection accuracy was always obtained by applying the large CNN models before NMS of the cascade output. Fourth, the embedding of existing detectors on the CompACT cascade can significantly enhance their

performance. The comparison between the columns of "Proposal" under "before NMS" and the baseline Faster-RCNN results in Table 5.4, shows that the proposal generation by CompACT is stronger than the innate proposal generation of the Faster-RCNN. In addition, the combination of "Embedded" and "before NMS" outperformed the baseline Faster-RCNN by 11.8, 5.58, and 4.26, points, for Alex, VGG, and ResNet, respectively! Finally, the overall best results were obtained by embedding the MS-CNN before NMS. Although the MS-CNN uses the backbone VGG-Net, it is better suited for multi-scale detection than the vanilla Faster-RCNN. Since pedestrians can appear at very diverse scales, its use leads to a better trade-off between the accuracy and complexity of the pedestrian detector. These observations show that many CNN detectors can be successfully embedded in CompACT, producing an embedded cascade of higher accuracy than *both* the original CompACT and the large CNN detectors.

Table 5.5 also confirms a property that we have observed very consistently in all our experiments: the complementarity between ACF and CNN features. This can, for example, be seen by comparing the performance of the single CNN feature cascade of Figure 5.5 (CONV5 19.96, CONV7 32.34, and CONV5CB 18.83) to the cascades using ACF+CNN features in Table 5.2 (Boosting 17.39 and CompACT 18.29). Note that, for the combination ACF+CNN, the CompACT cascade is not only faster (0.28 vs. 0.36, 0.37, 1.35) but also more accurate (18.29 vs. 19.96, 32.34, 18.83) than all single CNN feature cascades in Figure 5.5. More surprisingly, the ACF features are complementary even with large CNN features. This can be seen in Table 5.5, where *all* large CNN models achieve better performance when embedded in the CompACT cascade than when simply using the latter as a proposal mechanism.

With respect to detection complexity, Table 5.5 shows that there is no additional cost for "Embedded" over "Proposal". However, in both cases, the cost of embedding the R-CNN increases linearly with the number of proposals. This, and the fact that NMS rejects many overlapping proposals, explains why "before NMS" is much more expensive than "after NMS," for this model. On the other hand, because the Fast-RCNN computation is shared among proposal

regions, the complexity of embedding the Fast-RCNN is nearly constant, regardless of the number of proposals. These different settings make it possible to obtain multiple CompACT cascades of different detection accuracies and complexities. On the high-end of detection rates, the best trade-off between complexity and accuracy is achieved by the combination of Fast-RCNN, MS-CNN, "Embedded," and "before NMS." On the high-end of detection speeds, this holds for the combination of R-CNN, Alex, "Embedded," and "after NMS."

Finally, we note that the use of different embedding strategies and large CNN models enables a range of speed/accuracy trade-offs of interest for application developers. In this context, the flexibility enabled the family of models of Table 5.5 can far outweigh simple performance measures, such as speed or accuracy. By simply implementing CompACT, which does not use large CNN features, a developer can achieve a MR of 15.27 and a running time of 0.34. This is fairly cheap, and, as will be shown below, it is both faster and more accurate than other pedestrian detectors with this property. If higher accuracies are needed, the table offers a number of other choices. By augmenting CompACT with an R-CNN based on VGG-Net, "embedded" after NMS, accuracy can be increased by 3.9 points with a very marginal increase of 0.06s in running time. This requires a large CNN, but has virtually no additional cost in running time. It is a GPU based solution that emphasizes running time over accuracy. On the other hand, augmenting CompACT with a Fast-RCNN based on the MS-CNN, "embedded" before NMS, significantly increases accuracy (6.2 points) at the cost of a running time increase of 0.24s. This is one of the top detectors in the literature accuracy-wise, at the cost of a non-trivial increase in running time. The point is that the developer has a range of solutions to chose from, that cover a range of budgets in terms of accuracy, running time, energy consumption, and implementation cost.

5.5.7 Pedestrian Detection on Caltech

We next compared CompACT cascades to state of the art pedestrian detectors. We start by a comparison on Caltech, with the results of Figure 5.9. As usual, we present curves of miss-rate

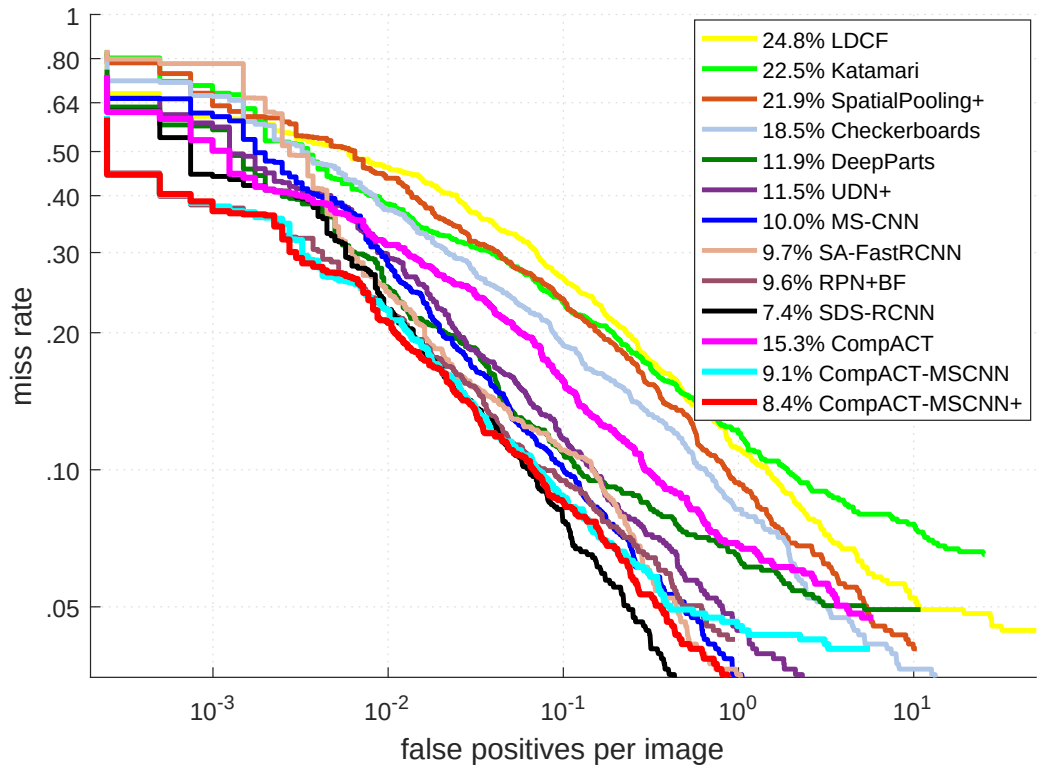


Figure 5.9: Comparison to the state-of-the-art on Caltech (reasonable).

v.s. false-positives per image (FPPI). CompACT uses ACF features and a small CNN, and outperforms all state of the art detectors of handcrafted features [192, 5, 127]. Note that some of these use many more features (HOG, LBP, spatial covariance, optical flow, multiple detectors, etc) and all are quite slow. On the other hand, CompACT runs at 3 fps on a relatively slow processor. CompACT-MSCNN embeds the MS-CNN in the last cascade stage. It outperforms most of the detectors using large CNNs, e.g. DeepParts [154], RPN+BF [190], SA-FastRCNN [92] and UDN+ [125]. The only competitive detector is the SDS-RCNN of [9]. This has a lower MR than CompACT-MSCNN, but the miss-rate v.s. FPPI curves behave very differently. CompACT-MSCNN dominates at low FPPI and SDS-RCNN at high FPPI. Overall, the two detectors are comparable. The comparison is also somewhat unfair because CompACT-MSCNN does not use a CNN to generate proposals. For completeness, we report on a modified version, denoted

Table 5.6: Pedestrian detection mAP comparison on KITTI.

Methods	Easy	Moderate	Hard	Time (s)
DPM	50.01	38.35	34.78	10
FilteredICF [192]	61.14	53.98	49.29	40
pAUCEnsT [127]	66.11	54.58	48.49	60
regionlet [164]	72.96	61.16	55.22	1
RCNN [64]	62.05	50.20	44.85	4
DeepParts [154]	70.46	58.68	52.73	1
CompACT	68.62	58.08	52.61	0.5
RPN+BF [190]	75.58	61.29	56.08	0.6
SDS-RCNN [9]	-	63.05	-	0.2
SDP+CRC [180]	77.81	64.25	59.31	0.6
Faster-RCNN [140]	78.35	65.91	61.19	2
Mono3D [15]	77.30	66.66	63.44	4.2
3DOP [16]	82.36	67.46	64.71	3
MM-MRFC [20]	82.37	69.96	64.76	0.05
F-PointNet [133]	87.81	77.25	74.46	0.17
CompACT-MSCNN	78.95	68.86	63.67	0.75

CompACT-MSCNN+, that combines the proposals generated by the MS-CNN and CompACT, and computes the final scores using the "Embedded" strategy. This improves performance at high FPPIs, which becomes close to that of SDS-RCNN.

5.5.8 Pedestrian Detection on KITTI

Table 5.6 compares the mAP of CompACT and CompACT-MSCNN to the state of the art on KITTI. Since KITTI test images are larger than those of Caltech, running times are higher for this dataset. Nevertheless, the CompACT cascade is among the fastest detectors. Among hand-crafted detectors (shown at the top of the table) only regionlet [164] has higher mAP. However, it uses very different features and is twice as slow. A more informative comparison is with channel based detectors, such as pAUCEnsT [127] and FilteredICF [192]. CompACT uses approximately the same number of feature channels (including the CNN model) but is both more accurate and faster. Again, CompACT even outperforms or is competitive with some detectors that use large CNNs, e.g. RCNN [64] and DeepParts [154].

Table 5.7: The results on CityPersons validation set.

Method	ACF	CompACT	Faster R-CNN	MS-CNN	CompACT-MSCNN+
MR	33.13	25.15	18.35	16.32	14.46

The CompACT-MSCNN cascade, which includes a large CNN, outperforms these methods by a large margin. The second part of the table compares it to models based on large CNNs. The table is ordered by increasing mAP in the moderate task. The detectors slightly faster than CompACT-MSCNN, RPN+BF [190], SDS-RCNN [9] and SDP+CRC [180], have substantially weaker mAP, e.g. a loss of about 4 mAP points for SDP+CRC. The next three detectors, Faster-RCNN, Mono3D [15] and 3DOP [16], have weaker mAP and speed than CompACT-MSCNN. For completeness, we also present the very best results on this dataset, i.e. MM-FRC [20] and F-PointNet [133]. These methods use stereo and LIDAR data, which the CompACT-MSCNN does not leverage. They show that 3D information is very helpful on KITTI. The design of cascades leveraging this information is left for future work. Note that the F-PointNet cannot be applied to color images alone, e.g. datasets like Caltech, and MM-MRFC has much weaker performance without 3D data, e.g. 12% on Caltech.

These results show that CompACT can generate high quality proposals even on datasets like KITTI, where proposal generation is known to be difficult [16]. It is particularly interesting that CompACT-MSCNN outperforms methods based on region proposal networks (RPN) [140], such as RPN+BF [190] or the Faster-RCNN. This shows that, using only small CNN models, CompACT can generate better proposals than the much more expensive RPN. Finally, it is also interesting that the closest competitors on Caltech, RPN+BF [190] and DeepParts [154], and the recent SDS-RCNN [9], are outperformed by CompACT-MSCNN on KITTI Moderate by a large margin: 7.57, 10.18 and 5.81 points, respectively.

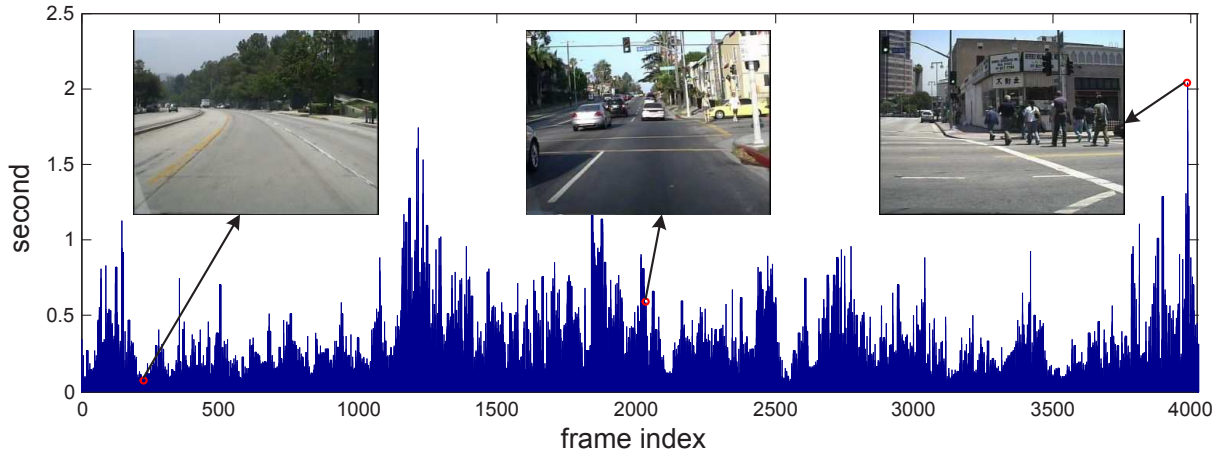


Figure 5.10: Processing time spent per Caltech test image.

5.5.9 Pedestrian Detection on CityPersons

CityPersons [193] is a recent pedestrian detection dataset, collected across multiple European cities. It has 2,975 training and 500 validation images, and 1,575 images for testing with held annotations. Images have size 2048×1024 and detection performance is evaluated as in Caltech. Table 5.7 compares the performace of the proposed detectors with the baselines of ACF [29] and Faster R-CNN [140], on the CityPersons validation set. For fair comparison, all detectors are implemented without upsampling the input images, and the Faster R-CNN was trained using the publicly available codebase of [10]. The CompACT cascade significantly outperforms ACF, and the CompACT-MSCNN+ cascade achieves better results than the Faster R-CNN. Note that the CompACT-MSCNN+ is an embedded cascade of higher accuracy than both the CompACT cascade and the very strong MS-CNN detector. These observations are consistent with the Caltech experiments.

5.5.10 CompACT as Attention Mechanism

One interesting property of CompACT, is that a detector cascade can be seen as an attention mechanism that assigns computation to image locations according to their detection

complexity. Figure 5.10 illustrates this property, summarizing the computation spent on each of the Caltech test images. Clearly, time complexity varies significantly with image content. The image on the left depicts a simple scene, with few objects and no pedestrian. Since simple and efficient features are enough to reject all windows, the image is processed with low complexity. The rightmost image depicts a more complicated scene with details of various scales (foreground vs. background buildings) and several pedestrians. In this case, many image regions propagate until the final stages of the cascade, and complexity is high. In this sense, CompACT behaves more like the human visual system. For example, a human annotator quickly realizes that the left image contains no pedestrians, but requires significantly longer time to count the number of pedestrians on the rightmost image.

This behavior is very different from channel-wise object detectors [192, 127] and CNN based object detectors [10, 190], whose time complexity is nearly independent of image content. From an application point of view, this can have benefits and disadvantages. For example, in Figure 5.10, processing time ranges from 0.05 to 2.0 seconds per frame, with a variance of about 0.04 seconds. This can create problems for applications that require a fixed processing time. For these, channel-wise or CNN detectors may be a better solution. On the other hand, there are benefits to the allocation of computation as needed, especially when power is an issue. For example, operating a Titan X GPU card 24-hours/day requires 6 kWh per day, i.e. 6 times the consumption of a fridge. This alone can preclude deployment on applications such as home surveillance, edge devices, drones, etc.

The ability of CompACT to detect the absence of pedestrians in a scene with simple CPU-based features enables drastic energy savings for applications on the home or edge, where such scenes can happen 99% of the time. On more complex and rare scenes the detector can activate energy intensive GPU features or even even ship the image regions in question to a central GPU, which serves all devices. Note that, on Caltech, the average image region size processed by a large CNN is 5.94% and 11.34% when embedding before and after NMS, respectively. This is

opposed to the 100% of standard CNN implementations, e.g. the Faster R-CNN [140]. Besides saving energy, this would split the large cost of a GPU by various edge devices, which could themselves be inexpensive.

5.6 Conclusion

In this work, we proposed the CompACT boosting algorithm for learning complexity-aware detector cascades. By optimizing classification risk under a complexity constraint, CompACT produces cascades that push features of high complexity to the later cascade stages. This has been shown to enable the seamless integration of multiple feature families in a unified design. This integration extends to features, such as deep CNNs, that were previously beyond the realm of cascaded detectors. The proposed CompACT cascades also generalize the popular combination of object proposals+CNN, which they were shown to outperform. Finally, we have shown that a pedestrian detector learned by application of CompACT to a diverse feature pool achieves accurate detection rates on Caltech and KITTI, at fairly fast speeds.

5.7 Acknowledgements

Chapter 5 is, in full, based on the material as it appears in the publication of "Learning Complexity-Aware Cascades for Pedestrian Detection", Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, to appear at *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2019. The dissertation author was the primary investigator and author of this material.

Chapter 6

Learning Low-Precision Neural Networks

6.1 Introduction

Deep neural networks have achieved state-of-the-art performance on computer vision problems, such as classification [86, 150, 152, 60, 61], detection [50, 140, 10], etc. However, their complexity is an impediment to widespread deployment in many applications of real world interest, where either memory or computational resource is limited. This is due to two main issues: large model sizes (50MB for GoogLeNet [152], 200M for ResNet-101 [61], 250MB for AlexNet [86], or 500M for VGG-Net [150]) and large computational cost, typically requiring GPU-based implementations. This generated interest in compressed models with smaller memory footprints and computation.

Several works have addressed the reduction of model size, through the use of quantization [21, 101, 96], low-rank matrix factorization [75, 28], pruning [55, 54], architecture design [97, 71], etc. Recently, it has been shown that weight compression by quantization can achieve very large memory savings, reducing each weight to as little as 1 bit, at a marginal cost in classification accuracy [21, 101]. It is, however, less effective along the computational dimension, because the core network operation, implemented by each of its units, is the dot-product between a weight and an activation vector. Complementing binary or quantized weights with quantized activations would enable the replacement of expensive dot-products by logical and bit-counting operations. Substantial speed ups should thus be possible if, in addition to weights, the inputs of each unit were binarized or quantized to low-bit.

However, the quantization of activations is more difficult than that of weights. For example, [22, 134] have shown that, while it is possible to binarize weights with a marginal cost in model accuracy, additional quantization of activations incurs nontrivial losses for large-scale classification, such as on ImageNet [143]. The difficulty is that binarization or quantization of activations requires their processing with non-differentiable operators that create problems for the backpropagation algorithm. This iterates between a feedforward step that computes

network outputs and a backpropagation step that computes the gradients required for learning. The difficulty is that binarization or quantization operators have step-wise responses that produce very weak gradient signals during backpropagation, compromising learning efficiency. So far, the problem has been addressed by using continuous approximations of the operator used in the feedforward step to implement the backpropagation step. This, however, creates a mismatch between the model that implements the forward computations and the derivatives used to learn it, leading to a sub-optimal model.

In this work, we view the quantization operator, used in the feedforward step, and the continuous approximation, used in the backpropagation step, as two functions that approximate the activation function of each network unit. We refer to these as the *forward* and *backward* approximation of the activation function. We start by considering the binary ± 1 quantizer, used in [22, 134], for which these two functions can be seen as a discrete and a continuous approximation of a non-linear activation function, the hyperbolic tangent, frequently used in classical neural networks. This activation is, however, not commonly used in recent deep learning literature, where the ReLU nonlinearity [117, 189, 60] has achieved much greater preponderance. This is exactly because it produces much stronger gradient magnitudes. While the hyperbolic tangent or sigmoid non-linearities are squashing non-linearities and mostly flat, the ReLU is an half-wave rectifier, of linear response to positive inputs. Hence, while the derivatives of the hyperbolic tangent are close to zero almost everywhere, the ReLU has unit derivative along the entire positive range of the axis.

To improve the learning efficiency of quantized networks, we consider the design of forward and backward approximation functions for the ReLU. To discretize its linear component, we propose to use an optimal quantizer. By exploiting the statistics of network activations and batch normalization operations that are commonly used in the literature, we show that this can be done with an half-wave Gaussian quantizer (HWGQ) that requires no learning and is very efficient to compute. While some recent works have attempted similar ideas [22, 134], their design of a

quantizer is not sufficient to guarantee good deep learning performance. We address this problem by complementing this design with a study of suitable backward approximation functions that account for the mismatch between the forward model and the back propagated derivatives. This study suggests operations such as linearization, gradient clipping or gradient suppression for the implementation of the backward approximation. We show that a combination of the forward HWGQ with these backward operations produces very efficient low-precision networks, denoted as HWGQ-Net, with much closer performance to continuous models, such as AlexNet [86], ResNet [61], GoogLeNet [152] and VGG-Net [150], than other available low-precision networks in the literature. To the best of our knowledge, this is the first time that a single low-precision algorithm could achieve success for so many popular networks. Using the arguments of [134], the HWGQ-Net (1-bit weights and 2-bit activations) could theoretically achieve $\sim$32$\times$ memory and $\sim$32$\times$ convolutional computation savings. This suggests that the HWGQ-Net can be very useful for the deployment of state-of-the-art neural networks in real world applications.

6.2 Related Work

The reduction of model size is a popular goal in the deep learning literature. One strategy is to exploit the widely known redundancy of neural network weights [27]. For example, [28, 75] proposed low-rank matrix factorization as a way to decompose a large weight matrix into several separable small matrices. An alternative procedure, known as connection pruning [55, 54], consists of removing unimportant connections of a pre-trained model and retraining, showing considerable model reduction without noticable loss in accuracy. Another model compression strategy is to constrain the model architecture itself, e.g. by removing fully connected layers, using convolutional filters of small size, etc. Many state-of-the-art deep networks, such as NIN [97], GoogLeNet [152] and ResNet [61], rely on such design choices. For example, SqueezeNet [71] has been shown to achieve a parameter reduction of $\sim$50 times, for accuracy comparable to

that of AlexNet. Moreover, hash functions have also been used to compress model size [14].

Another branch of approaches for model compression is weight binarization [21, 134, 22] or quantization [101, 96, 52]. [159] used a fixed-point representation to quantize weights of pre-trained neural networks. [52] showed that vector quantization enables 4$\sim$8 times compression with minimal accuracy loss. [96] proposed a method for fixed-point quantization based on the optimal bit-width allocations across network layers. [90, 101] have shown that ternary weight quantization into levels $\{-1,0,1\}$ can achieve 16$\times$ or 32$\times$ model compression with slight accuracy loss, even on large-scale classification tasks. Finally, [21] has shown that filter weights can be quantized to ± 1 without noticeable loss of classification accuracy on CIFAR-10 [85].

Since quantization of activations enables further speed-ups and reduces training memory requirements, it has attracted some attention [159, 96, 22, 134, 197, 95, 101]. [159, 96] performed the quantization after network training, avoiding the issues of nondifferentiable optimization. Recently, [22, 134, 197] tried to tackle the nondifferentiable optimization issue by using a continuous approximation to the quantizer function in the backpropagation step. [95] proposed several potential solutions to the problem of gradient mismatch and [101, 197] showed that gradients can be quantized with a small number of bits during the backpropagation step. While some of these methods have produced good results on CIFAR-10, none has produced low precision networks competitive with full-precision models on large-scale classification tasks, such as ImageNet [143].

6.3 Binary Networks

We start with a brief review of the issues involved in the binarization of a deep network.

6.3.1 Goals

Deep neural networks consist of layers of units that roughly model the computations of neurons in the mammalian brain. Each unit computes an activation function

$$z = g(\mathbf{w}^T \mathbf{x}), \tag{6.1}$$

where $\mathbf{w} \in \mathbb{R}^{c \cdot w \cdot h}$ is a weight vector, $\mathbf{x} \in \mathbb{R}^{c \cdot w \cdot h}$ an input vector, and $g(\cdot)$ a non-linear function. A convolutional network implements layers of these units, where weights are usually represented as a tensor $\mathbf{W} \in \mathbb{R}^{c \times w \times h}$. The dimensions c, w and h are defined by the number of filter channels, width and height, respectively. Since modern networks have very large numbers of these units, the structure of (6.1) is the main factor in the complexity of the overall model. This complexity can be a problem for applications along two dimensions. The first is the large memory footprint required to store weights $\mathbf{w}$. The second is the computational complexity required to compute large numbers of dot-products $\mathbf{w}^T \mathbf{x}$. Both difficulties are compounded by the requirement of floating point storage of weights and floating point arithmetic to compute dot-products, which are not practical for many applications. This has motivated interest in low-precision networks [22, 134, 197].

6.3.2 Weight Binarization

An effective strategy to binarize the weights $\mathbf{W}$ of convolutional filters, which we adopt in this work, has been proposed by [134]. This consists of approximating the full precision weight matrix $\mathbf{W}$, used to compute the activations of (6.1) for all the units, by the product of a binary matrix $\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$ and a scaling factor $\alpha \in \mathbb{R}^+$, such that $\mathbf{W} \approx \alpha \mathbf{B}$. A convolutional operation on input $\mathbf{I}$ can then be approximated by

$$\mathbf{I} * \mathbf{W} \approx \alpha(\mathbf{I} \oplus \mathbf{B}), \tag{6.2}$$

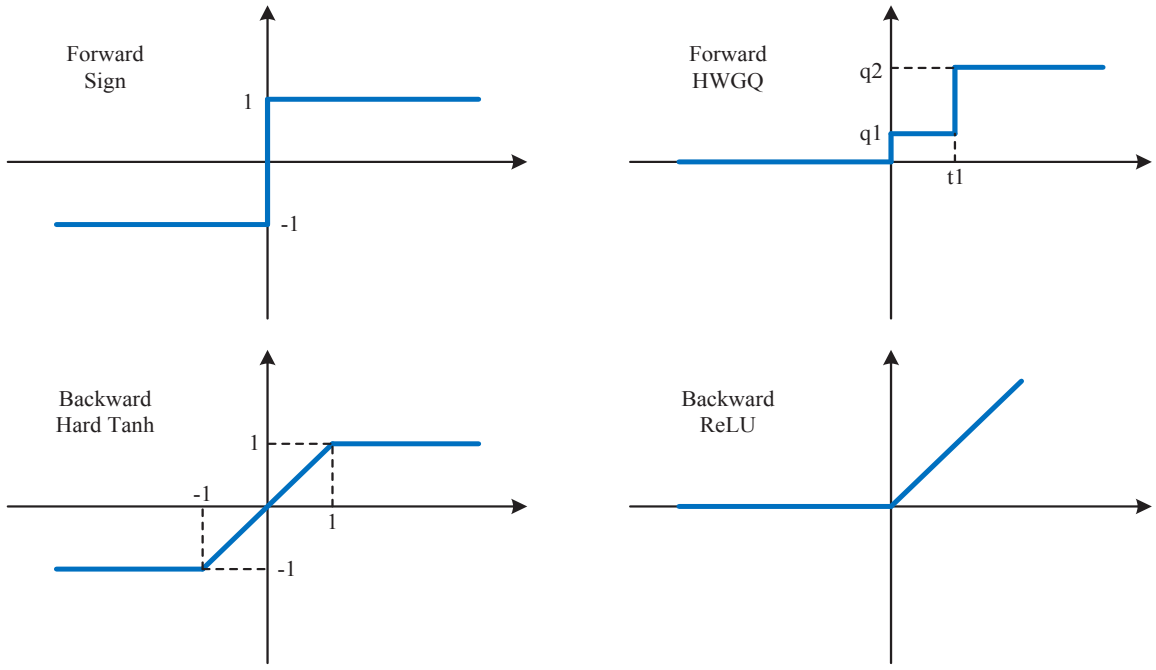


Figure 6.1: Forward and backward functions for binary *sign* (left) and half-wave Gaussian quantization (right) activations.

where $\oplus$ denotes a multiplication-free convolution. [134] has shown that an optimal approximation can be achieved with $\mathbf{B}^* = sign(\mathbf{W})$ and $\alpha^* = \frac{1}{cwh}\|\mathbf{W}\|_1$. While binary weights tremendously reduce the memory footprint of the model, they do not fully solve the problem of computational complexity. Substantial further reductions of complexity can be obtained by the binarization of $\mathbf{I}$, which enables the implementation of dot products in (6.2) with logical and bit-counting operations [22, 134].

6.3.3 Binary Activation Quantization

The use of binary activations has been suggested in [134, 22, 197]. It is usually implemented by replacing $g(x)$ in (6.1) with the *sign* non-linearity

$$z = sign(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{otherwise} \end{cases} \tag{6.3}$$

shown in Figure 6.1. This creates difficulties to the backpropagation algorithm used to learn the neural network, by minimizing a cost C with respect to the weights $\mathbf{w}$. Consider the unit of (6.1). The derivative of C with respect to $\mathbf{w}$ is

$$\frac{\partial C}{\partial \mathbf{w}} = \frac{\partial C}{\partial z} g'(\mathbf{w}^T \mathbf{x})\mathbf{x}. \tag{6.4}$$

When $g(x)$ is replaced by (6.3), the derivative $g'(\mathbf{w}^T \mathbf{x})$ is zero almost everywhere and the gradient magnitudes tend to be very small. In result, the gradient descent algorithm does not converge to minima of the cost. To overcome this problem, [22] proposed to use an alternative function, *hard tanh*, which we denote by $\widetilde{sign}$, in the backpropagation step. This function is shown in Figure 6.1, and has derivative

$$\widetilde{sign}'(x) = \begin{cases} 1, & \text{if } |x| \leq 1 \\ 0, & \text{otherwise.} \end{cases} \tag{6.5}$$

In this work, we denote (6.3) as the forward and (6.5) as the backward approximations of the activation non-linearity $g(x)$ of (6.1). These approximations have two main problems. The first is that they approximate the hyperbolic tangent (tanh), which is a squashing non-linearity. The saturating behavior of squashing non-linearities (such as the tanh or the sigmoid) emphasizes the problem of vanishing derivatives, compromising the effectiveness of backpropagation. The second is that the discrepancy between the approximation of $g(x)$ by the forward *sign* and by the backward $\widetilde{sign}$ creates a mismatch between the feedforward model and the derivatives used to learn it. In result, backpropagation can be highly suboptimal. This is called the "gradient mismatch" problem [95].

6.4 Half-wave Gaussian Quantization

In this section, we propose an alternative quantization strategy, the approximation of the ReLU non-linearity.

6.4.1 ReLU

The ReLU is the half-wave rectifier defined by [117],

$$g(x) = \max(0, x). \tag{6.6}$$

It is now well known that, when compared to squashing non-linearities, its use in (6.1) significantly improves the efficiency of the backpropagation algorithm. It thus seems more sensible to rely on ReLU approximations for network quantization than those of the previous section. We propose a quantizer $Q(x)$ to approximate (6.6) in the feedforward step and a suitable piecewise linear approximation $\widetilde{Q}(x)$ for the backpropagation step.

6.4.2 Forward Approximation

A quantizer is a piecewise constant function

$$Q(x) = q_i, \quad if \quad x \in (t_i, t_{i+1}], \tag{6.7}$$

that maps all values of x within quantization interval $(t_i, t_{i+1}]$ into a quantization level $q_i \in \mathbb{R}$, for $i = 1, \cdots, m$. In general, $t_1 = -\infty$ and $t_{m+1} = \infty$. This generalizes the *sign* function, which can be seen as a 1-bit quantizer. A quantizer is denoted uniform if

$$q_{i+1} - q_i = \Delta, \quad \forall i, \tag{6.8}$$

where Δ is a constant quantization step. The quantization levels q_i act as reconstruction values for x, under the constraint of reduced precision. Since, for any x, it suffices to store the quantization index i of (6.7) to recover the quantization level q_i, non-uniform quantization requires $\log_2 m$ bits of storage per activation x. However, more than $\log_2 m$ bits are needed to represent x in

arithmetic operations, since these use q_i, not the index i. For a uniform quantizer, where Δ is a universal scaling factor that can be placed in evidence, $\log_2 m$ bits are enough for both storage and arithmetic computation.

Optimal quantizers are usually defined in the mean-squared error sense, i.e.

$$\begin{aligned} Q^*(x) &= \arg\min_Q E_x[(Q(x)-x)^2] \\ &= \arg\min_Q \int p(x)(Q(x)-x)^2 dx \end{aligned} \tag{6.9}$$

where $p(x)$ is the probability density function of x. Hence, the optimal quantizer of the dot-products of (6.1) depends on their statistics. While the optimal solution $Q^*(x)$ of (6.9) is usually non-uniform, a uniform solution $Q^*(x)$ is available by adding the uniform constraint of (6.8) to (6.9). Given dot product samples, the optimal solution of (6.9) can be obtained by Lloyd's algorithm [106]. This, however, is an iterative algorithm. Since a different quantizer must be designed per network unit, and this quantizer changes with the backpropagation iteration, the straightforward application of this procedure is computationally intractable.

This difficulty can be avoided by exploiting the statistical structure of the activations of deep networks. For example, [70, 73] have noted that the dot-products of (6.1) tend to have a symmetric, non-sparse distribution, that is close to Gaussian. Taking into account the fact that the ReLU is a half-wave rectifier, this suggests the use of the half-wave Gaussian quantizer (HWGQ),

$$Q(x) = \begin{cases} q_i, & \text{if } x \in (t_i, t_{i+1}], \\ 0, & x \leq 0, \end{cases} \tag{6.10}$$

where $q_i \in \mathbb{R}^+$ for $i = 1, \cdots, m$ and $t_i \in \mathbb{R}^+$ for $i = 1, \cdots, m+1$ ($t_1 = 0$ and $t_{m+1} = \infty$) are the optimal quantization parameters for the Gaussian distribution. The adoption of the HWGQ guarantees that these parameters only depend on the mean and variance of the dot-product distribution. However, because these can vary across units, it does not eliminate the need for the

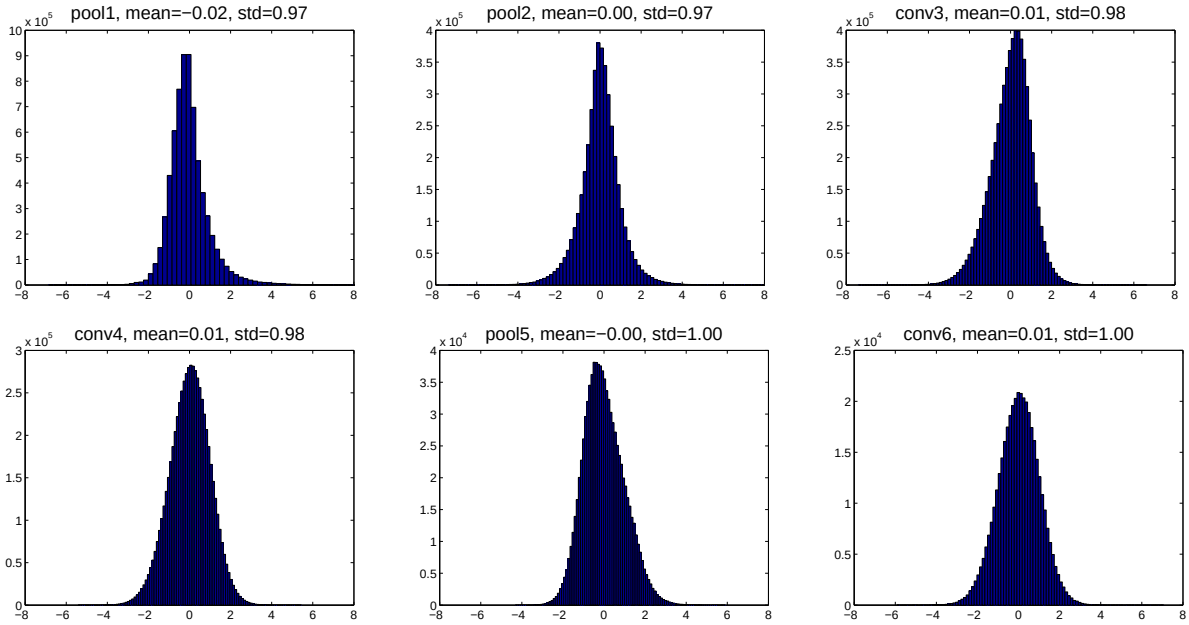


Figure 6.2: Dot-product distributions on different layers of AlexNet with binary weights and quantized activations (100 random images).

repeated application of Lloyd's algorithm across the network.

This problem can be alleviated by resorting to batch normalization [73]. This is a widely used normalization technique, which forces the responses of each network layer to have zero mean and unit variance. We apply this normalization to the dot-products, with the result illustrated in Figure 6.2, for a number of AlexNet units of different layers. Although the distributions are not perfectly Gaussian and there are minor differences between them, they are all close to Gaussian with zero mean and unit variance. It follows that the optimal quantization parameters q_i^* and t_i^* are approximately identical across units, layers and backpropagation iterations. Hence, Lloyd's algorithm can be applied once, with data from the entire network. In fact, because all distributions are approximately Gaussian of zero mean and unit variance, the quantizer can even be designed from samples of this distribution. In our implementation, we drew 10^6 samples from a standard Gaussian distribution of zero mean and unit variance, and obtained the optimal quantization parameters by Lloyd's algorithm. The resulting parameters t_i^* and q_i^* were used to parametrize a single HWGQ that was used in all layers, after batch normalization of dot-products.

6.4.3 Backward Approximation

Since the HWGQ is a step-wise constant function, it has zero derivative almost everywhere. Hence, the approximation of $g(x)$ by $Q(x)$ in (6.4) leads to the problem of vanishing derivatives. As in Section 6.3, a piecewise linear function $\widetilde{Q}(x)$ can be used during the backpropagation step to avoid weak convergence. In summary, we seek a piece-wise function that provides a good approximation to the ReLU and to the HWGQ. We next consider three possibilities.

Vanilla ReLU: Since the ReLU of (6.6) is already a piece-wise linear function, it seems sensible to use the ReLU itself, denoted the *vanilla ReLU*, as the backward approximation function. This corresponds to using the derivative

$$\widetilde{Q}'(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

in (6.4). The forward and backward approximations $Q(x)$ and $\widetilde{Q}(x)$ of the ReLU are shown in Figure 6.1. Note that, while the backward approximation is exact, it is not equal to the forward approximation. Hence, there is a gradient mismatch. For $x > 0$, the approximation of $Q(x)$ by the ReLU has error $|Q(x) - x|$. This is upper bounded by $(t_{i+1} - q_i)$ for $x \in (t_i, t_{i+1}]$ but unbounded when $x \in (t_m, \infty)$. Hence, the mismatch is particularly large for large values of x. Since these are the values on the tail of the distribution of x, the ReLU is said to have a large mismatch with $Q(x)$ "on the tail". When the ReLU is used to approximate $g'(x)$ in (6.4), it can originate very inaccurate gradients for large dot-products. In our experience, this can make the learning algorithm unstable.

This is a classical problem in the robust estimation literature, where outliers can unduly influence the performance of a learning algorithm [69]. For quantization, where $Q(x)$ assumes that values of x beyond q_m have very low probability, large dot-products are effectively outliers. The classical strategy for outlier mitigation is to limit the growth rate of the error function, in this case $|Q(x) - x|$. Hence, the problem is the monotonicity of the ReLU beyond $x = q_m$. To address

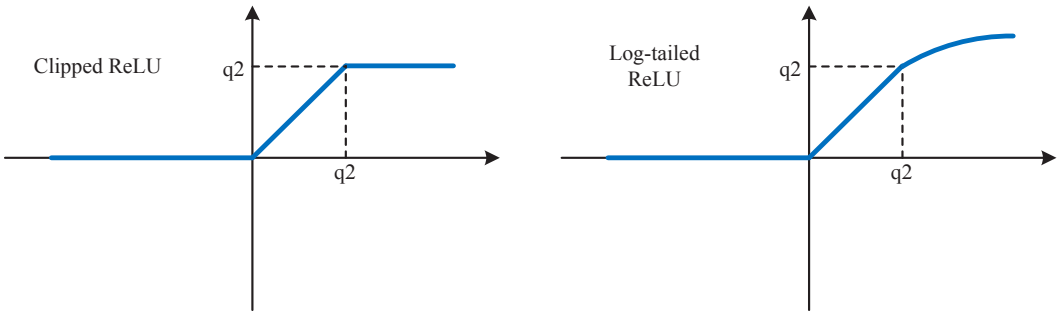


Figure 6.3: Backward piece-wise activation functions of clipped ReLU and log-tailed ReLU.

it, we investigate alternative backwards approximation functions of slower growth rate.

Clipped ReLU: The first approximation, denoted the *clipped ReLU*, is identical to the vanilla ReLU in $(-\infty, q_m]$ but constant beyond $x = q_m$,

$$\widetilde{Q}_c(x) = \begin{cases} q_m, & x > q_m, \\ x, & x \in (0, q_m], \\ 0, & \text{otherwise.} \end{cases} \tag{6.12}$$

Its use to approximate $g'(\mathbf{w}^T\mathbf{x})$ in (6.4) guarantees that there is no mismatch on the tail. Gradients are non-zero only for dot-products in the interval $(0, q_m]$. As illustrated in Figure 6.3, the clipped ReLU is a better match for the HWGQ than the vanilla ReLU. In our experiments, ReLU clipping was proved very useful to guarantee a stable optimization. This is similar to previous observations that gradient clipping robustifies the learning of very deep networks [129].

Log-tailed ReLU: Ideally, a network with quantized activations should approach the performance of a full-precision network as the number of quantization levels m increases. The sensitivity of the vanilla ReLU approximation to outliers limits the performance of low precision networks. While the clipped ReLU alleviates this problem, it can impair network performance due to the loss of information in the clipped interval (q_m, ∞). An intermediate solution is to use, in this interval, a function whose growth rate is in between that of the clipped ReLU (zero derivative) and the ReLU (unit derivative). One possibility is to enforce logarithmic growth on

the tail, according to

$$\widetilde{Q}_l(x) = \begin{cases} q_m + \log(x-\tau), & x > q_m, \\ x, & x \in (0, q_m], \\ 0, & x \leq 0, \end{cases} \tag{6.13}$$

where $\tau = q_m - 1$. This is denoted the *log-tailed ReLU* and is shown in Figure 6.3. It has derivative

$$\widetilde{Q}_l'(x) = \begin{cases} 1/(x-\tau), & x > q_m, \\ 1, & x \in (0, q_m], \\ 0, & x \leq 0. \end{cases} \tag{6.14}$$

When used to approximate $g'(x)$ in (6.4), the log-tailed ReLU is identical to the vanilla ReLU for amplitudes smaller than q_m, but gives decreasing weight to amplitudes larger than this. It behaves like the vanilla ReLU (unit derivative) for $0 < x \leq q_m$ but converges to the clipped ReLU (zero derivative) as x grows to infinity.

6.5 Experimental Results

The proposed HWGQ-Net was evaluated on ImageNet (ILSVRC2012) [143], which has $\sim$1.2M training images from 1K categories and 50K validation images. The evaluation metrics were top-1 and top-5 classification accuracy. Several popular networks were tested: AlexNet [86], ResNet [61], a variant of VGG-Net [150, 60], and GoogLeNet [152]. Our implementation uses Caffe [76], see source code at https://github.com/zhaoweicai/hwgq.

6.5.1 Implementation Details

In all experiments, training images were resized to 256×256, and a 224×224 (227×227 for AlexNet) crop was randomly sampled from an image or its horizontal flip. Batch normalization

[73] was applied before each quantization layer, as discussed in Section 6.4.2. The ratio of dropout [62] was set as 0.1 for networks with binary weights and full activations, but no dropout was used for networks with quantized activations. All networks were learned from scratch with SGD. No data augmentation was used other than standard random image flipping and cropping. No bias term was used for binarized weights. As in [134], networks with quantized activations used max-pooling before batch normalization. This is denoted "layer re-ordering". As in [134, 197], the first and last network layers had full precision. Evaluation was based solely on central 224×224 crops.

On AlexNet [86] experiments, the mini-batch size was 256, weight decay 0.0005, and learning rate started at 0.01. For ResNet, we used the parameters of [61]. For the variant of VGG-Net, denoted VGG-Variant, a smaller version of model-A in [60], only 3 convolutional layers were used for input size of 56, 28 and 14, and the "spp" layer was removed. The mini-batch size was 128, and learning rate started at 0.01. For GoogLeNet [152], the branches for side losses were removed, in the inception layers, max-pooling was removed and the channel number of the "reduce" 1×1 convolutional layers was increased to that of their following 3×3 and 5×5 convolutional layers. Weight decay was 0.0002 and the learning strategy was similar to ResNet [61]. For all networks tested, momentum was 0.9, and when mini-batch size was 256 (128), the learning rate was divided by 10 after every 50K (100K) iterations, 160K (320K) in total. Only AlexNet, ResNet-18 and VGG-Variant were explored in the following ablation studies. In all tables and figures, "FW" indicates full-precision weights, "BW" binary weights, and "Full" full-precision weights and activations.

6.5.2 Full-precision Activation Comparison

Before considering the performance of the forward quantized activation functions $sign(x)$ and $Q(x)$, we compared the performance of the continuous $\widetilde{sign}(x)$ (hard tanh) and $\widetilde{Q}(x)$ (ReLU) as activation function. In this case, there is no activation quantization nor forward/backward

Table 6.1: Full-precision Activation Comparison for AlexNet.

	Full	FW+$\widetilde{sign}$	FW+$\widetilde{Q}$	BW+$\widetilde{sign}$	BW+$\widetilde{Q}$
Top-1	55.7	46.7	55.7	43.9	53.9
Top-5	79.3	71.0	79.3	68.3	77.3

Table 6.2: Low-bit Activation Comparison.

Model		Full	BW	FW+Q	BW+$sign$	BW+Q
AlexNet	Top-1	55.7	52.4	49.5	39.5	46.8
	Top-5	79.3	75.9	73.7	63.6	71.0
ResNet-18	Top-1	66.3	61.3	37.5	42.1	33.0
	Top-5	87.5	83.6	61.9	67.1	56.9
VGG-Variant	Top-1	68.6	65.5	48.3	50.1	44.1
	Top-5	88.9	86.5	72.3	74.3	68.7

gradient mismatch. AlexNet results are presented in Table 6.1, using identical settings for $\widetilde{sign}(x)$ and $\widetilde{Q}(x)$, for fair comparison. As expected from the discussion of Sections 6.3 and 6.4, $\widetilde{Q}(x)$ achieved substantially better performance than $\widetilde{sign}(x)$, for both FW and BW networks. The fact that these results upper bound the performance achievable when quantization is included suggests that $sign(x)$ is not a good choice for quantization function. $Q(x)$, on the other hand, has a fairly reasonable upper bound.

6.5.3 Low-bit Activation Quantization Results

We next compared the performance achieved by adding the *sign* and HWGQ $Q(x)$ (backward vanilla ReLU) quantizers to the set-up of the previous section. The results of AlexNet, ResNet-18 and VGG-Variant are summarized in Table 6.2. Notice, first, that BW has weaker performance than BW+Q of AlexNet in Table 6.1, due to the impact of the layer re-ordering [134] introduced in Section 6.5.1. Next, comparing BW to FW+Q, where the former binarizes weights only and the latter quantizes activations only, it can be seen that weight binarization causes a minor degradation of accuracy. This is consistent with the findings of [134, 22]. On the other hand, activation quantization leads to a nontrivial loss. This confirms that the latter is a more

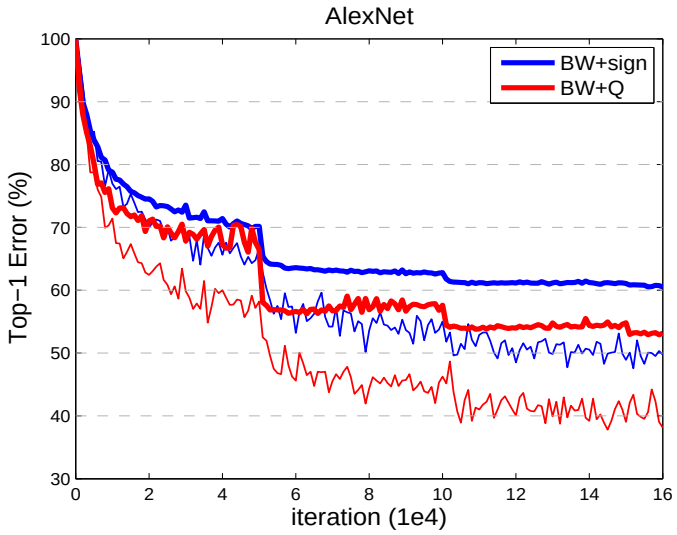


Figure 6.4: The error curves of training (thin) and test (thick) for $sign(x)$ and $Q(x)$ (HWGQ) activation functions.

difficult problem.

When weight binarization and activation quantization were combined, recognition performance dropped even further. For AlexNet, the drop was much more drastic for BW+*sign* (backward hard tanh) than for BW+Q (backward vanilla ReLU). These results support the hypotheses of Section 6.3 and 6.4, as well as the findings of Table 6.1. The training errors of BW+*sign* and BW+Q for AlexNet are shown in Figure 6.4. Note the much lower training error of $Q(x)$, suggesting that it enables a much better approximation of the full precision activations than $sign(x)$. Nevertheless, the gradient mismatch due to the use of $Q(x)$ as forward and the vanilla ReLU as backward approximators made the optimization somewhat instable. For example, the error curve of BW+Q is bumpy during training. This problem becomes more serious for deeper networks. In fact, for the ResNet-18 and VGG-Variant, BW+Q performed worse than BW+*sign*. This can be explained by the fact that the *sign* has a smaller gradient mismatch problem than the vanilla ReLU. Substantial improvements are possible by correcting the mismatch between the forward quantizer $Q(x)$ and its backward approximator.

Table 6.3: Backward Approximations Comparison.

Model		BW	no-opt	vanilla	clipped	log-tailed
AlexNet	Top-1	52.4	30.0	46.8	48.6	49.0
	Top-5	75.9	53.6	71.0	72.8	73.1
ResNet-18	Top-1	61.3	34.2	33.0	54.5	53.5
	Top-5	83.6	59.6	56.9	78.5	77.7
VGG-Variant	Top-1	65.5	42.8	44.1	60.9	60.6
	Top-5	86.5	68.3	68.7	83.2	82.9

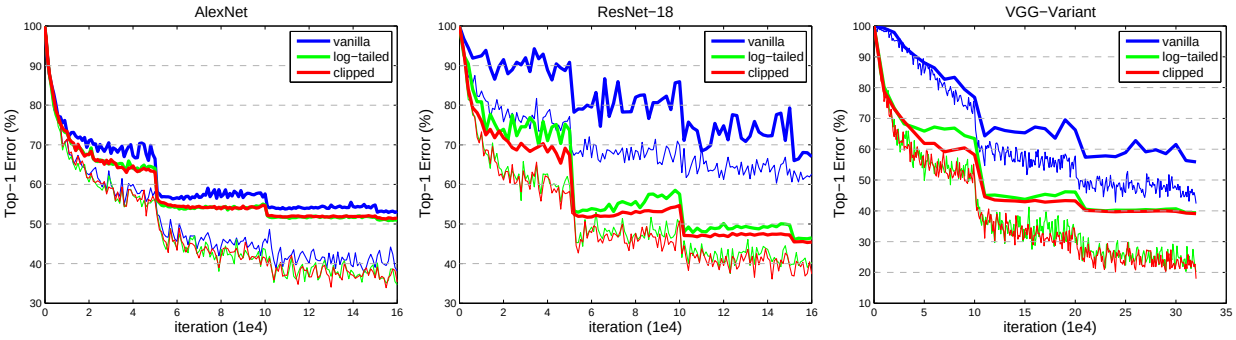


Figure 6.5: The error curves of training (thin) and test (thick) for alternative backward approximations.

6.5.4 Backward Approximations Comparison

We next considered the impact of the backward approximators of Section 6.4.3. Table 6.3 shows the performance under the different approximations. In all cases, weights were binarized and the HWGQ was used as forward approximator (quantizer). "no-opt" refers to the quantization of activations of pre-trained BW networks. This requires no nondifferentiable approximation, but fails to account for the quantization error. We attempted to minimize the impact of cumulative errors across the network by recomputing the means and variances of all batch normalization layers. Even after this, "no-opt" had significantly lower accuracy than the full-precision activation networks.

Substantial gains were obtained by training the activation quantized networks from scratch. Although the vanilla ReLU had reasonable performance as backwards approximator for AlexNet, much better results were achieved with the clipped ReLU of (6.12) and the log-tailed ReLU

Table 6.4: Bit-width Comparison of Activation Quantization.

quantization type		non-uniform				uniform		none
# levels		2	3	7	15	3*	7*	BW
AlexNet	Top-1	48.6	50.6	52.4	52.6	50.5	51.9	52.4
	Top-5	72.8	74.3	75.8	76.2	74.6	75.7	75.9
ResNet-18	Top-1	54.5	57.6	60.3	60.8	56.1	59.6	61.3
	Top-5	78.5	81.0	82.8	83.4	79.7	82.4	83.6

of (6.13). Figure 6.5 shows that the larger gradient mismatch of the vanilla ReLU created instabilities in the optimization, for all networks. However, these instabilities were more serious for the deeper networks, such as ResNet-18 and VGG-Variant. This explains the sharper drop in performance of the vanilla ReLU for these networks, in Table 6.3. Note, in Figure 6.5, that the clipped ReLU and the log-tailed ReLU enabled more stable learning and reached a much better optimum for all networks. Among them, the log-tailed ReLU performed slightly better than the clipped ReLU on AlexNet, but slightly worse on ResNet-18 and VGG-Variant. To be consistent, "clipped ReLU" was used in the following sections.

6.5.5 Bit-width Impact

The next set of experiments studied the bit-width impact of the activation quantization. In all cases, weights were binarized. Table 6.4 summarizes the performance of AlexNet and ResNet-18 as a function of the number of quantization levels. While the former improved with the latter, there was a saturation effect. The default HWGQ configuration, also used in all previous experiments, consisted of two non-uniform positive quantization levels plus a "0". This is denoted as "2" in the table. For AlexNet, this very low-bit quantization sufficed to achieve recognition rates close to those of the full-precision activations. For this network, quantization with seven non-uniform levels was sufficient to reproduce the performance of full-precision activations. For ResNet-18, however, there was a more noticeable gap between low-bit and full-precision activations. These results suggest that increasing the number of quantization levels is more

Table 6.5: HWGQ implementation of various popular networks.

Model		Reference	Full	HWGQ
AlexNet	Top-1	57.1	58.5	52.7
	Top-5	80.2	81.5	76.3
ResNet-18	Top-1	69.6	67.3	59.6
	Top-5	89.2	87.9	82.2
ResNet-34	Top-1	73.3	69.4	64.3
	Top-5	91.3	89.1	85.7
ResNet-50	Top-1	76.0	71.5	64.6
	Top-5	93.0	90.5	85.9
VGG-Variant	Top-1	-	69.8	64.1
	Top-5	-	89.3	85.6
GoogLeNet	Top-1	68.7	71.4	63.0
	Top-5	88.9	90.5	84.9

beneficial for ResNet-18 than for AlexNet.

Table 6.4 also shows the results obtained with uniform quantization, with superscript "$*$". Interestingly, for the same number of quantization levels, the performance of the uniform quantizer was only slightly worse than that of its non-uniform counterpart. This is, however, not a completely fair comparison since, as discussed in Section 6.4.2, non-uniform quantization requires more bits for arithmetic operations. For the same bit width, e.g. both "2" and "3*" require a 2-bit representation for arithmetic computation, the uniform quantizer was noticeably superior to the non-uniform quantizer.

6.5.6 Comparison to the state-of-the-art

Table 6.5[1] presents a comparison between the full precision and the low-precision HWGQ-Net of many popular network architectures. In all cases, the HWGQ-Net used 1-bit binary weights, a 2-bit uniform HWGQ as forward approximator, and the clipped ReLU as backwards approximator. Comparing to the previous ablation experiments, the numbers of training iterations

[1]The reference performance of AlexNet and GoogLeNet is at https://github.com/BVLC/caffe, and of ResNet is at https://github.com/facebook/fb.resnet.torch. Our worse ResNet implementations are probably due to fewer training iterations and no further data augmentation.

Table 6.6: Comparison to state-of-the-art low-precision methods. Top-1 gap to the corresponding full-precision network is also reported.

Model	AlexNet			ResNet-18	
	XNOR	DOREFA	HWGQ	XNOR	HWGQ
Top-1	44.2	47.7	52.7	51.2	59.6
Top-5	69.2	-	76.3	73.2	82.2
Top-1 gap	-12.4	-8.2	-5.8	-18.1	-7.7

were doubled and polynomial learning rate annealing (power of 1) was used for HWGQ-Net. Table 6.5 shows that the HWGQ-Net approximates well all popular networks, independently of their complexity or depth. The top-1 accuracy drops from full- to low-precision are similar for all networks (5$\sim$9 points), suggesting that low-precision HWGQ-Net will achieve improved performance as better full-precision networks become available.

Training a network with binary weights and low-precision activations from scratch is a new and challenging problem, only addressed by a few previous works [22, 134, 197]. Table 6.6 compares the HWGQ-Net with the recent XNOR-Net [134] and DOREFA-Net [197], on the ImageNet classification task. The DOREFA-Net result is for a model of binary weights, 2-bit activation, full precision gradient and no pre-training. For AlexNet, the HWGQ-Net outperformed the XNOR-Net and the DOREFA-Net by a large margin. Similar improvements over the XNOR-Net were observed for the ResNet-18, where DOREFA-Net results are not available. It is worth noting that the gaps between the full-precision networks and the HWGQ-Net (-5.8 for AlexNet and -7.7 for ResNet-18) are much smaller than those of the XNOR-Net (-12.4 for AlexNet and -18.1 for ResNet-18) and the DOREFA-Net (-8.2 for AlexNet). This is strong evidence that the HWGQ is a better activation quantizer. Note that, in contrast to the experimentation with one or two networks by [22, 134, 197], the HWGQ-Net is shown to perform well for various network architectures. To the best of our knowledge, this is the first time that a single low-precision network is shown to successfully approximate many popular networks.

Table 6.7: The results on CIFAR-10. The bit width before and after "+" is for weights and activations respectively.

precision	Method	error (%)
Full + Full	Maxout [53]	9.38
	NIN [97]	8.81
	DSN [88]	8.22
	FitNet [141]	8.39
	ResNet-110 [61]	6.43
	VGG-Small	6.82
1-bit + Full	BinaryConnect [21]	8.27
2-bit + Full	Ternary Weight Network [90]	7.44
1-bit + 1-bit	BNN [22]	10.15
1-bit + 2-bit	VGG-Small-HWGQ	7.49

6.5.7 Results on CIFAR-10

In addition, we conducted some experiments on the CIFAR-10 dataset [85]. The network structure used, denoted VGG-Small, was similar to that of [22] but relied on a cross-entropy loss and without two fully connected layers. The learning strategy was that used in the VGG-Variant of Section 6.5.1, but the mini-batch size was 100 and the learning rate was divided by 10 after every 40K iterations (100K in total). The data augmentation strategy of [61] was used. As in Section 6.5.6, polynomial learning rate decay was used for low-precision VGG-Small. The HWGQ-Net results are compared with the state-of-the-art in Table 6.7. It can be seen that the HWGQ-Net achieved better performance than various popular full-precision networks, e.g. Maxout [53], NIN [97], DSN [88], FitNet [141], and than various low precision networks. The low-precision VGG-Small drops 0.67 points when compared to its full-precision counterpart. These findings are consistent with those on ImageNet.

6.6 Conclusion

In this work, we considered the problem of training high performance deep networks with low-precision. This was achieved by designing two approximators for the ReLU non-linearity:

a half-wave Gaussian quantizer in the feedforward computations, and a piece-wise continuous function in the backpropagation step. This design overcomes the learning inefficiency of the popular binary *sign* quantization procedure. To minimize the problem of gradient mismatch, we have studied several backwards approximation functions, including clipped ReLU and log tailed ReLU approximators. The proposed network, denoted HWGQ-Net, was shown to significantly outperform previous efforts at deep learning with low precision, for various state-of-the-art networks. These promising results suggest that the HWGQ-Net can be very useful for the deployment of state-of-the-art neural networks in real world applications.

6.7 Acknowledgements

Chapter 6 is, in full, based on the material as it appears in the publication of "Deep Learning with Low Precision by Half-wave Gaussian Quantization", Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos, In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2017. The dissertation author was the primary investigator and author of this material.

Chapter 7

Discussion and Conclusion

In this thesis, we have studied the challenges on the universality of object detection, including scale-variance, high-quality requirement, domain shift, hardware and computational constraints, etc. To address each of the challenge, we have proposed a series of novel solutions. At first, we built an effective and efficient multi-scale object detection architecture for deep neural network, which uses feature pyramid as an approximation to image pyramid. This new design not only achieved the state-of-the-art scale-invariant detection performance, but also enabled real-time running speeds. Second, we discussed why high-quality object detection is important but widely neglected by the community, and then we proposed a novel multi-stage architecture (Cascade R-CNN) for high-quality detection. The Cascade R-CNN has shown to be very effective when the detection quality requirement is high. Next, we studied another important subfield in object detection, multi-domain and domain-universal object detection, and proposed a couple of solutions to them. They can substantially extend the detector universality over domains, and even sometimes surpass the domain specialized object detector. Finally, to address the challenges induced by hardware and computational constraints, we proposed solutions from two different perspectives. The former one is on the design of complexity-aware cascades, which is able to push the expensive cascade stages to the end of the cascade detector, where fewer hypotheses are left to classify. This enables to learn the optimal cascade configurations given the computational constraints, e.g. CPU-only or mixed CPU/GPU solutions. The latter one is on low-precision networks, which could significantly reduce the computation from the full-precision networks. Our novel low-precision networks achieve very close performance to the full-precision counterparts. All these efforts have substantially advanced the universality of object detection.

Beyond the above aspects, some other aspects are also important for object detection universality, e.g. video, 3D and point cloud data, the transferability to downstream tasks, etc. First, we live in the world with a dimension of time and many detection applications are for video platform, but the current detectors are widely developed for images. It is more sensible to exploit the temporal information and structure to achieve better and more stable object detector.

One difficulty is the expensive cost to annotate the video data, which could be orders of higher than that of image annotation. However, there is a large amount of unlabeled videos, and the temporal structure can be exploited to enable weak/self-supervised learning from videos. Second, we also live in the world with a dimension of depth, which is broadly neglected in 2D object detection for 2D images. 3D vision is better aligned with human vision system, and 3D object detection could help to better interact with the physical world. Another special format of 3D data is point cloud, which is important and popular for autonomous driving and domestic robots. It has sparse data representation, very different from the typical image dense representation. Its unique properties could derive very different architectures for feature representation learning and efficient point cloud detection system. Third, as we discussed before, object detection has many downstream tasks, e.g. visual question answering, captioning, visual navigation, robot grasping, knowledge graph, pose estimation, etc. The current object detectors are frequently trained without knowing that it is going to be used in other tasks, and their transferability could vary application by application. It will be interesting to develop a task-universal object detector, which could be well transferred to a large number of various downstream tasks. And this task-universal object detector could save the repeating development costs for each individual downstream task, and will have broader impact.

Object detection has been playing an important role in our everyday life. Although it has made huge progresses in the past a few years and, in this thesis, we have advanced its universality from multiple different aspects, there is still a long journey to go to achieve the ultimate goal: a universal object detector that can work in any case under any condition like human visual system. These will motivate me to continue to work on object detection and try to push forward its limits in the future.

Bibliography

[1] P. A. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marqués, and J. Malik. Multiscale combinatorial grouping. In *CVPR*, pages 328–335, 2014.

[2] S. Bell, C. L. Zitnick, K. Bala, and R. B. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016.

[3] R. Benenson, M. Mathias, R. Timofte, and L. J. V. Gool. Pedestrian detection at 100 frames per second. In *CVPR*, pages 2903–2910, 2012.

[4] R. Benenson, M. Mathias, T. Tuytelaars, and L. J. V. Gool. Seeking the strongest rigid detector. In *CVPR*, pages 3666–3673, 2013.

[5] R. Benenson, M. Omran, J. Hosang, , and B. Schiele. Ten years of pedestrian detection, what have we learned? In *ECCV, CVRSUAD workshop*, 2014.

[6] H. Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

[7] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms - improving object detection with one line of code. In *ICCV*, pages 5562–5570, 2017.

[8] L. D. Bourdev and J. Brandt. Robust object detection via soft cascade. In *CVPR*, pages 236–243, 2005.

[9] G. Brazil, X. Yin, and X. Liu. Illuminating pedestrians via simultaneous detection and segmentation. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 4960–4969, 2017.

[10] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, pages 354–370, 2016.

[11] Z. Cai, M. J. Saberian, and N. Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *ICCV*, pages 3361–3369, 2015.

[12] Z. Cai and N. Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018.

[13] X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. In *CVPR*, pages 2887–2894, 2012.

[14] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015.

[15] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, pages 2147–2156, 2016.

[16] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, pages 424–432, 2015.

[17] M. Cheng, Z. Zhang, W. Lin, and P. H. S. Torr. BING: binarized normed gradients for objectness estimation at 300fps. In *CVPR*, pages 3286–3293, 2014.

[18] C. Cortes, M. Mohri, and U. Syed. Deep boosting. In *ICML*, pages 1179–1187, 2014.

[19] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[20] A. D. Costea, R. Varga, and S. Nedevschi. Fast boosting based detection using scale invariant multimodal multiresolution filtered features. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 993–1002, 2017.

[21] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[22] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[23] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, pages 3150–3158, 2016.

[24] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. In *NIPS*, pages 379–387, 2016.

[25] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *ICCV*, 2017.

[26] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005.

[27] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *NIPS*, pages 2148–2156, 2013.

[28] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277, 2014.

[29] P. Dollár, R. Appel, S. J. Belongie, and P. Perona. Fast feature pyramids for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(8):1532–1545, 2014.

[30] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, pages 1–11, 2009.

[31] P. Dollár, P. Welinder, and P. Perona. Cascaded pose regression. In *CVPR*, pages 1078–1085, 2010.

[32] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, 2012.

[33] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, pages 1841–1848, 2013.

[34] M. Dredze, A. Kulesza, and K. Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 79(1-2):123–149, 2010.

[35] C. Elkan. The foundations of cost-sensitive learning. In *IJCAI*, pages 973–978, 2001.

[36] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[37] T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6(Apr):615–637, 2005.

[38] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[39] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[40] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, pages 23–37, 1995.

[41] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[42] J. Gall and V. S. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, pages 1022–1029, 2009.

[43] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, pages 3354–3361, 2012.

[44] A. Geiger, C. Wojek, and R. Urtasun. Joint 3d estimation of objects and scene layout. In *NIPS*, pages 1467–1475, 2011.

[45] G. Georgakis, M. A. Reza, A. Mousavian, P.-H. Le, and J. Kosecka. Multiview rgb-d dataset for object instance detection. *arXiv preprint arXiv:1609.07826*, 2016.

[46] S. Gidaris and N. Komodakis. Object detection via a multi-region and semantic segmentation-aware CNN model. In *ICCV*, pages 1134–1142, 2015.

[47] S. Gidaris and N. Komodakis. Attend refine repeat: Active box proposal generation via in-out localization. In *BMVC*, 2016.

[48] S. Gidaris and N. Komodakis. Locnet: Improving localization accuracy for object detection. In *CVPR*, pages 789–798, 2016.

[49] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[50] R. B. Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448, 2015.

[51] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.

[52] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

[53] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML*, pages 1319–1327, 2013.

[54] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[55] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[56] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, 2012.

[57] K. He, R. Girshick, and P. Dollár. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, 2018.

[58] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, 2017.

[59] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, pages 346–361, 2014.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[61] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[62] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[63] J. Hosang, R. Benenson, P. Dollár, and B. Schiele. What makes for effective detection proposals? *PAMI*, 2015.

[64] J. H. Hosang, M. Omran, R. Benenson, and B. Schiele. Taking a deeper look at pedestrians. In *CVPR*, 2015.

[65] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei. Relation networks for object detection. In *IEEE CVPR*, volume 2, 2018.

[66] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.

[67] P. Hu and D. Ramanan. Finding tiny faces. In *CVPR*, pages 1522–1530, 2017.

[68] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

[69] P. J. Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[70] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[71] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[72] N. Inoue, R. Furuta, T. Yamasaki, and K. Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. In *CVPR*, pages 5001–5009, 2018.

[73] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[74] L. Itti and P. Baldi. A principled approach to detecting surprising events in video. In *CVPR*, volume 1, pages 631–637, 2005.

[75] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[76] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM*, pages 675–678, 2014.

[77] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang. Acquisition of localization confidence for accurate object detection. In *ECCV*, pages 816–832, 2018.

[78] W. Jiang, E. Zavesky, S.-F. Chang, and A. Loui. Cross-domain learning methods for high-level visual concept classification. In *ICIP*, pages 161–164. IEEE, 2008.

[79] M. Joshi, W. W. Cohen, M. Dredze, and C. P. Rosé. Multi-domain learning: when do domains matter? In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1302–1312. Association for Computational Linguistics, 2012.

[80] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[81] T. Kato, H. Kashima, M. Sugiyama, and K. Asai. Multi-task learning via conic programming. In *NeurIPS*, pages 737–744, 2008.

[82] A. Khosla, T. Zhou, T. Malisiewicz, A. A. Efros, and A. Torralba. Undoing the damage of dataset bias. In *ECCV*, pages 158–171. Springer, 2012.

[83] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017.

[84] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, page 8, 2017.

[85] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[86] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[87] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. In *ECCV*, pages 765–781, 2018.

[88] C. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[89] B. Li, T. Wu, and S. Zhu. Integrating context and occlusion for car detection by hierarchical and-or model. In *ECCV*, pages 652–667, 2014.

[90] F. Li and B. Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

[91] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *CVPR*, pages 5325–5334, 2015.

[92] J. Li, X. Liang, S. Shen, T. Xu, and S. Yan. Scale-aware fast R-CNN for pedestrian detection. *CoRR*, abs/1510.08160, 2015.

[93] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Detnet: Design backbone for object detection. In *ECCV*, pages 339–354, 2018.

[94] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, pages 3158–3165, 2013.

[95] D. D. Lin and S. S. Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *CoRR*, abs/1607.02241, 2016.

[96] D. D. Lin, S. S. Talathi, and V. S. Annapureddy. Fixed point quantization of deep convolutional networks. In *ICML*, pages 2849–2858, 2016.

[97] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[98] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *ECCV*, pages 740–755, 2014.

[99] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[100] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[101] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *CoRR*, abs/1510.03009, 2015.

[102] A. Liu, Y. Su, W. Nie, and M. S. Kankanhalli. Hierarchical clustering multi-task learning for joint human action grouping and recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(1):102–114, 2017.

[103] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *IEEE CVPR*, pages 8759–8768, 2018.

[104] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *ECCV*, pages 21–37, 2016.

[105] W. Liu, S. Liao, W. Hu, X. Liang, and X. Chen. Learning efficient single-stage pedestrian detectors by asymptotic localization fitting. In *ECCV*, pages 643–659, 2018.

[106] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Information Theory*, 28(2):129–136, 1982.

[107] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[108] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

[109] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[110] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, pages 67–82, 2018.

[111] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. MIT Press, 2010.

[112] H. Masnadi-Shirazi and N. Vasconcelos. High detection-rate cascades for real-time object detection. In *ICCV*, pages 1–6, 2007.

[113] H. Masnadi-Shirazi and N. Vasconcelos. Cost-sensitive boosting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(2):294–309, 2011.

[114] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *NIPS*, pages 512–518, 1999.

[115] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, pages 3994–4003, 2016.

[116] A. Møgelmose, M. M. Trivedi, and T. B. Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Trans. Intelligent Transportation Systems*, 13(4):1484–1497, 2012.

[117] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[118] M. Najibi, M. Rastegari, and L. S. Davis. G-CNN: an iterative grid based object detector. In *CVPR*, pages 2369–2377, 2016.

[119] M. Najibi, P. Samangouei, R. Chellappa, and L. S. Davis. Ssh: Single stage headless face detector. In *ICCV*, pages 4885–4894, 2017.

[120] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, pages 4293–4302, 2016.

[121] W. Nam, P. Dollár, and J. H. Han. Local decorrelation for improved pedestrian detection. In *NIPS*, pages 424–432, 2014.

[122] E. Ohn-Bar and M. M. Trivedi. Learning to detect vehicles by clustering appearance patterns. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2511–2521, 2015.

[123] W. Ouyang, K. Wang, X. Zhu, and X. Wang. Learning chained deep features and classifiers for cascade in object detection. *CoRR*, abs/1702.07054, 2017.

[124] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *ICCV*, pages 2056–2063, 2013.

[125] W. Ouyang, H. Zhou, H. Li, Q. Li, J. Yan, and X. Wang. Jointly learning deep features, deformable parts, occlusion and classification for pedestrian detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(8):1874–1887, 2018.

[126] S. Paisitkriangkrai, C. Shen, and A. van den Hengel. Pedestrian detection with spatially pooled features and structured ensemble learning. *CoRR*, abs/1409.5209, 2014.

[127] S. Paisitkriangkrai, C. Shen, and A. van den Hengel. Strengthening the effectiveness of pedestrian detection with spatially pooled features. In *ECCV*, pages 546–561, 2014.

[128] S. E. Palmer. *Vision science: Photons to phenomenology*. MIT press, 1999.

[129] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318, 2013.

[130] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE signal processing magazine*, 32(3):53–69, 2015.

[131] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. In *IEEE CVPR*, pages 6181–6189, 2018.

[132] B. Pepik, M. Stark, P. V. Gehler, and B. Schiele. Multi-view and 3d deformable part models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(11):2232–2245, 2015.

[133] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. pages 918–927, 2018.

[134] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, 2016.

[135] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, pages 506–516, 2017.

[136] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *CVPR*, pages 8119–8127, 2018.

[137] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016.

[138] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[139] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1137–1149, 2017.

[140] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[141] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014.

[142] A. Rosenfeld and J. K. Tsotsos. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[143] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[144] M. J. Saberian and N. Vasconcelos. Learning optimal embedded cascades. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(10):2005–2018, 2012.

[145] M. J. Saberian and N. Vasconcelos. Boosting algorithms for detector cascade learning. *Journal of Machine Learning Research*, 15(1):2569–2605, 2014.

[146] R. E. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML*, pages 322–330, 1997.

[147] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, pages 3626–3633, 2013.

[148] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *CVPR*, 2007.

[149] A. Shrivastava, A. Gupta, and R. B. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, pages 761–769, 2016.

[150] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[151] B. Singh and L. S. Davis. An analysis of scale invariance in object detection–snip. In *IEEE CVPR*, pages 3578–3587, 2018.

[152] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[153] D. Tang, Y. Liu, and T. Kim. Fast pedestrian detection by cascaded random forest with dominant orientation templates. In *BMVC*, pages 1–11, 2012.

[154] Y. Tian, P. Luo, X. Wang, and X. Tang. Deep learning strong parts for pedestrian detection. In *ICCV*, pages 1904–1912, 2015.

[155] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. In *CVPR*, page 4, 2017.

[156] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[157] P. P. Vaidyanathan. *Multirate systems and filter banks*. Pearson Education India, 1993.

[158] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders. Segmentation as selective search for object recognition. In *ICCV*, pages 1879–1886, 2011.

[159] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[160] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[161] P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[162] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille. Towards unified depth and semantic prediction from a single image. In *CVPR*, pages 2800–2809, 2015.

[163] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *CVPR*, 2018.

[164] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, pages 17–24, 2013.

[165] J. M. Wolfe. Visual attention. In *Seeing*, pages 335–386. Elsevier, 2000.

[166] X. Wu, D. Zhang, J. Zhu, and S. C. H. Hoi. Single-shot bidirectional pyramid networks for high-quality object detection. *CoRR*, abs/1803.08208, 2018.

[167] Y. Wu and K. He. Group normalization. In *ECCV*, pages 3–19, 2018.

[168] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. Dota: A large-scale dataset for object detection in aerial images. In *Proc. CVPR*, 2018.

[169] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Data-driven 3d voxel patterns for object category recognition. In *CVPR*, pages 1903–1911, 2015.

[170] R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. In *ICCV*, pages 1–8, 2007.

[171] R. Xiao, L. Zhu, and H. Zhang. Boosting chain learning for object detection. In *ICCV*, pages 709–715, 2003.

[172] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995, 2017.

[173] S. Xie and Z. Tu. Holistically-nested edge detection. In *ICCV*, pages 1395–1403, 2015.

[174] X. Xiong and F. D. la Torre. Supervised descent method and its applications to face alignment. In *CVPR*, pages 532–539, 2013.

[175] J. Yan, Z. Lei, D. Yi, and S. Li. Learn to combine multiple hypotheses for accurate face alignment. In *ICCV Workshops*, pages 392–396, 2013.

[176] K. Yan, M. Bagheri, and R. M. Summers. 3d context enhanced region-based convolutional neural network for end-to-end lesion detection. In *ICCV*, pages 511–519, 2018.

[177] K. Yan, X. Wang, L. Lu, L. Zhang, A. Harrison, M. Bagheri, and R. M. Summers. Deep lesion graphs in the wild: relationship learning and organization of significant radiology image findings in a diverse large-scale lesion database. In *IEEE CVPR*, 2018.

[178] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Convolutional channel features. In *ICCV*, pages 82–90, 2015.

[179] B. Yang, J. Yan, Z. Lei, and S. Z. Li. CRAFT objects from images. In *CVPR*, pages 6043–6051, 2016.

[180] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *CVPR*, pages 2129–2137, 2016.

[181] J. Yang, J. Lu, D. Batra, and D. Parikh. A faster pytorch implementation of faster r-cnn. *https://github.com/jwyang/faster-rcnn.pytorch*, 2017.

[182] S. Yang, P. Luo, C. C. Loy, and X. Tang. WIDER FACE: A face detection benchmark. In *CVPR*, pages 5525–5533, 2016.

[183] Y. Yang and T. M. Hospedales. A unified perspective on multi-domain and multi-task learning. *arXiv preprint arXiv:1412.7489*, 2014.

[184] S. Yantis. Control of visual attention. *attention*, 1(1):223–256, 1998.

[185] A. L. Yarbus. Eye movements during perception of complex objects. In *Eye movements and vision*, pages 171–211. Springer, 1967.

[186] D. Yoo, S. Park, J. Lee, A. S. Paek, and I. Kweon. Attentionnet: Aggregating weak directions for accurate object detection. In *ICCV*, pages 2659–2667, 2015.

[187] S. Zagoruyko, A. Lerer, T. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A multipath network for object detection. In *BMVC*, 2016.

[188] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *CVPR*, pages 3712–3722, 2018.

[189] M. D. Zeiler, M. Ranzato, R. Monga, M. Z. Mao, K. Yang, Q. V. Le, P. Nguyen, A. W. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. On rectified linear units for speech processing. In *ICASSP*, pages 3517–3521, 2013.

[190] L. Zhang, L. Lin, X. Liang, and K. He. Is faster R-CNN doing well for pedestrian detection? In *ECCV*, pages 443–457, 2016.

[191] S. Zhang, C. Bauckhage, and A. B. Cremers. Informed haar-like features improve pedestrian detection. In *CVPR*, pages 947–954, 2014.

[192] S. Zhang, R. Benenson, and B. Schiele. Filtered channel features for pedestrian detection. In *CVPR*, pages 1751–1760, 2015.

[193] S. Zhang, R. Benenson, and B. Schiele. Citypersons: A diverse dataset for pedestrian detection. In *CVPR*, pages 4457–4465, 2017.

[194] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-shot refinement neural network for object detection. In *IEEE CVPR*, 2018.

[195] Y. Zhang and Q. Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2017.

[196] W. Zheng and L. Liang. Fast car detection using image strip features. In *CVPR*, pages 2703–2710, 2009.

[197] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

[198] Y. Zhu, R. Urtasun, R. Salakhutdinov, and S. Fidler. segdeepm: Exploiting segmentation and context in deep neural networks for object detection. In *CVPR*, pages 4703–4711, 2015.

[199] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, pages 391–405, 2014.