

UC Merced

UC Merced Electronic Theses and Dissertations

Title

Learning to Detect in Images and Videos

Permalink

<https://escholarship.org/uc/item/94z2v66n>

Author

Hsu, Han-Kai

Publication Date

2018

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, MERCED

Learning to Detect in Images and Videos

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Electrical Engineering and Computer Science

by

Han-Kai Hsu

Committee in charge:

Professor Ming-Hsuan Yang, Chair
Professor Shawn Newsam
Professor Sungjin Im

2018

Copyright
Han-Kai Hsu, 2018
All rights reserved.

The thesis of Han-Kai Hsu is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Professor Shawn Newsam

Professor Sungjin Im

Professor Ming-Hsuan Yang

Chair

University of California, Merced

2018

TABLE OF CONTENTS

	Signature Page	iii
	Table of Contents	iv
	List of Figures	vi
	List of Tables	x
	Vita and Publications	xi
	Abstract	xii
Chapter 1	Introduction	1
Chapter 2	Learning to Tell Brake and Turn Signals in Videos	4
	2.1 Introduction	4
	2.2 Related Work	5
	2.3 Proposed Algorithm	7
	2.3.1 Rear Signal Recognition System	8
	2.3.2 Convolution Neural Network	10
	2.3.3 Long Short-Term Memory	11
	2.4 Experimental results	13
	2.4.1 Data pre-processing	14
	2.4.2 Network settings	14
	2.4.3 Performance evaluation	15
	2.5 Summary	17
Chapter 3	Learning from Photoshop Operation Videos	19
	3.1 Introduction	19
	3.2 Dataset Construction Procedure	21
	3.3 Dataset Description.	24
	3.4 Tasks and Evaluation	27
	3.5 Methodology	29
	3.5.1 Attention-aware Filtering	30
	3.5.2 Video Regularization	32
	3.5.3 3-D CNN	32
	3.5.4 Data Augmentation	33
	3.6 Experiments	34
	3.6.1 Ablation Study	34
	3.6.2 Analysis on the Command Classification Task	35
	3.7 Summary	37

Chapter 4	Progressive Domain Adaptation for Object Detection	38
4.1	Introduction	38
4.2	Related Work	41
4.3	Progressive Domain Adaptation	42
4.3.1	Adaptation in the Feature Space	43
4.3.2	Progressive Adaptation	45
4.4	Algorithm Analysis	49
4.4.1	Stage-one Analysis	49
4.4.2	Stage-two Analysis	50
4.5	Experimental Results	51
4.5.1	Implementation Details	52
4.5.2	Datasets	52
4.5.3	Cross Camera Adaptation	53
4.5.4	Weather Adaptation	56
4.5.5	Adaptation to Large-scale Dataset	56
4.5.6	Synthetic Images and Visual Results	58
4.6	Summary	59
Chapter 5	Conclusion and Future Work	63
Bibliography		65

LIST OF FIGURES

Figure 2.1:	The brake and turn signal classifiers utilized in this system are based on a CNN-LSTM structure. The proposed system integrates the two classifiers with a 16-frame sequence as the input and outputs the rear light labels represented by three letters: B (brake), L (left) and R (right). We receive O (off) when the corresponding signal is turned off. RGB frames are passed to the brake classifier to obtain the first letter of the sequence state. Then, the computed difference image of the sequence is cropped into two ROI regions as input for the turn classifier and outputs the last two letters of the sequence state.	6
Figure 2.2:	Examples of rear signal states.	7
Figure 2.3:	Computing the difference between two consecutive frames. (a) Frame from previous step X_{i-1} . (b) Current frame X_i . (c) The computed SIFT flow [53] between X_{i-1} and X_i . (d) W_i is the warped image of X_i using SIFT flow in (c). (e) Absolute difference of W_i and X_{i-1} .	9
Figure 2.4:	ROI is indicated as the blue highlighted area, where the left and right blue regions are propagated through the network separately.	10
Figure 2.5:	Illustration of our network architecture. The layers before fc6 serve to extract spatial information from our input. Afterwards, we send the features to the LSTM module to utilize the temporal information and further classify the desired state in the last fully-connected layer.	11
Figure 2.6:	The LSTM unit in the proposed algorithm.	13
Figure 2.7:	Brake and turn failure cases. (a) and (b) are brake recognition failures affected by strong sunlight and is difficult for human to recognize the signal state. (c) is a turn failure case caused by sunlight and shades which in result acts similar to flashing signals.	17
Figure 3.1:	Example frames of the PSOV dataset. Each row represents a video clip for one specific Photoshop operation.	20
Figure 3.2:	Labeling tool for Photoshop videos. This figure shows an example interface of our online labeling tool, by which workers can easily annotate a Photoshop operation video. Past annotations are listed on the left. See detailed descriptions in Section 3.2.	23
Figure 3.3:	Pipeline of dataset construction procedure.	24

Figure 3.4:	Data distribution in the PSOV dataset. From left to right, we show data distribution of all labeled commands and the top-50. The first row shows the number of each command, and the second row shows the average duration (finish time minus start time) of the commands, all sorted by value. The blue lines in duration figures denote duration variance values of each bar. The top-left figure shows that about 150 commands only have 1 labeled sequence, and that the top-50 commands have the number of sequences more than 100. We provide labeling data for all command sequences in the dataset, but only evaluate tasks on the top-50 to guarantee enough training data.	25
Figure 3.5:	Dataset structure. This figure illustrates the data storage path in the dataset, where folder <i>video</i> , <i>commandLabels</i> , <i>segments</i> , <i>metadata</i> contains whole videos, per-video annotation file, operation frames, and video caption information respectively.	26
Figure 3.6:	Key difficulties in the PSOV dataset. This figure shows examples of three key difficulties in the PSOV dataset, i.e. duration variance, tiny motion and background clutter. 1) For time variance, each row shows a sequence of class <i>File</i> > <i>New</i> . This same operation has a two-second difference in the two sequences. 2) For tiny movement, optical flow [22] is used to illustrate what can be seen in motion space: the brush moving around lower eyelid in the first sequence is too weak to be detected; and the new layer appeared in the right panel has no stronger response than background noises. 3) For background clutter, we show an extreme example where the center region is zoomed in by the creator, largely changing the background of the operation.	27
Figure 3.7:	Attention-aware filtering. We show the attention filtering results on example frames from two commands: <i>Layer Panel</i> > <i>Select Layer</i> and <i>Layer Panel</i> > <i>Duplicate Layer</i> . $\overrightarrow{Mask}$ denotes the attention computing in single direction ($frm_{t-1} \Rightarrow frm_t$, $frm_t \Rightarrow frm_{t+1}$); and $\overleftrightarrow{Mask}$ denotes the result considering bi-directional context ($frm_{t-1} \Rightarrow frm_t \Leftarrow frm_{t+1}$).	30
Figure 3.8:	Network structure.	32
Figure 3.9:	Accuracy distribution. Image (a) shows the per-class accuracy distribution on training set class duration variance; image (b) shows the per-sequence prediction versus averaged attention filter area curve. See details in Section 3.6.2.	35

Figure 3.10:	Confusion matrix for Difference Filtering and Attention-aware Filtering. Index of each row and column denotes a top-50 class; color blue to red in each pixel indicates the proportion of its row-class classified into column-class; class 0 and 12 are <i>Layer Panel > Select Layer</i> and <i>Layer Panel > Duplicate Layer</i> respectively.	36
Figure 4.1:	An illustration of our progressive adaptation method. Conventional domain adaptation aims to solve domain-shift problem from source to target domain, which is denoted as $l_{S \rightarrow T}$. We propose to bridge this gap with an intermediate synthetic domain that allows us to gradually solve separate subtasks with smaller gaps (shown as $l_{S \rightarrow F}$ and $l_{F \rightarrow T}$). In addition, we treat each image in the synthetic domain unequally based on its quality with respect to the target domain, where the larger size in yellow triangle stands for larger weights (i.e., the closer to the target, the higher of the weight).	39
Figure 4.2:	The overall structure of our adaptation network. Labeled and unlabeled images are both passed through the encoder network E to extract CNN features $feat_L$ and $feat_U$. We then use them to: 1) learn supervised object detection with the detector network from $feat_L$, and 2) forward both features to GRL and a domain discriminator, learning domain-invariant features in an adversarial manner. Our progressive adaptation framework is shown in Figure 4.3. . . .	43
Figure 4.3:	The proposed progressive adaptation framework. The algorithm includes two stages of adaptation as shown in a) and b). In a), we first transform source images to generate synthetic ones by using the generator G learned via CycleGAN [99]. Afterward, we use the labeled source domain and perform first stage adaptation to the synthetic domain. Then in b), our model applies a second stage adaptation which takes the synthetic domain with labels inherited from the source and aligns the synthetic domain features with the target distribution. In addition, a weight w is obtained from the discriminator D_{cycle} in CycleGAN to balance the synthetic image qualities in the detection loss. The adaptation network is shown in Figure 4.2.	46
Figure 4.4:	Visualization of the feature distributions via t-SNE [89], showing that our synthetic images serve as an intermediate feature space between the source and target distributions. Each dot represents one image feature extracted from E . We take 500 images from the Cityscapes validation set and 500 from the KITTI training set for comparison.	48

Figure 4.5:	Image quality examples from the KITTI dataset synthesized to be in the Cityscapes domain. a) shows the ones that are translated with better quality. Images in b) contain artifacts and fail to preserve details of the car, almost blending into the background.	48
Figure 4.6:	Histogram plot of the weight distribution applied to the task loss during the second stage of our method for KITTI $\rightarrow$ Cityscapes. The x-axis is the weight and the y-axis is the number of images. . .	51
Figure 4.7:	Examples of the source and synthetic images used in KITTI $\rightarrow$ Cityscapes. Top row is the KITTI images while the bottom row is the synthetic images after being transformed.	58
Figure 4.8:	Examples of the source and synthetic images used in Cityscapes $\rightarrow$ Foggy Cityscapes. Top row is the Cityscapes images while the bottom row is the synthetic images after being transformed.	59
Figure 4.9:	Examples of the source and synthetic images used in Cityscapes $\rightarrow$ BDD100k daytime. Top row is the Cityscapes images while the bottom row is the synthetic images after being transformed.	59
Figure 4.10:	Examples of the detection results for the task KITTI $\rightarrow$ Cityscapes. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels. .	60
Figure 4.11:	Examples of the detection results for the task Cityscapes $\rightarrow$ Foggy Cityscapes. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels.	61
Figure 4.12:	Examples of the detection results for the task Cityscapes $\rightarrow$ BDD100k daytime. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels.	62

LIST OF TABLES

Table 2.1:	Data set for turn and brake lights.	14
Table 2.2:	Brake recognition accuracy (%) comparison with using the CNN and our method (CNN-LSTM). The results are based on five-fold cross validation.	14
Table 2.3:	Turn recognition component analysis of the proposed algorithm with different combinations of features and modules. RGB/difference image (Diff), ROI, and LSTM are added for evaluation based on prediction accuracy (%).	15
Table 2.4:	Turn recognition comparison with using the CNN and our method (CNN-LSTM). Both methods use difference image and ROI for inputs, and evaluated using five-fold cross validation.	15
Table 2.5:	Final recognition accuracy (%) evaluated using five-fold cross validation.	16
Table 3.1:	Ablation study. This table shows the performance for different combinations of components in the proposed framework, where 3D-Conv, Diff-Filter, Bi-Context, DataAug denote 3-D Convolution, Difference Filtering, Bi-direction Context, Data Augmentation respectively (details in Section 3.6.1).	33
Table 4.1:	Performance of the first-stage adaptation. For each scenario, we show the mean average precision (mAP) of the synthetic and target domains, before and after applying our first-stage adaptation.	50
Table 4.2:	Performance comparison during the second stage on KITTI $\rightarrow$ Cityscapes using different data.	51
Table 4.3:	Cross camera adaptation using KITTI and Cityscapes datasets. The results show the average precision (AP) of the <i>car</i> class shared between the two domains.	54
Table 4.4:	Analysis of our weighted task loss compared to several arbitrary weight settings. We show that by setting each image weight with respect to the distance from the target distribution improves the model performance.	54
Table 4.5:	Weather adaptation focusing on clear weather to foggy weather using the Cityscapes and Foggy Cityscapes datasets respectively. Performance is evaluated using the mean average precision (mAP) across 8 classes.	55
Table 4.6:	Adaptation from a smaller Cityscapes dataset to a larger and diverse BDD100k dataset. A subset of the BDD100k dataset labeled as <i>daytime</i> is used as the target domain. We evaluate the mean average precision (mAP) of 10 classes which are available across the two domains.	57

VITA

- 2014 B. E. in Technology Application and Human Resource Development, National Taiwan Normal University, Taipei, Taiwan
- 2018 M. S. in Electrical Engineering and Computer Science, University of California, Merced

PUBLICATIONS

Jingchun Cheng*, Han-Kai Hsu*, Fang Chen, Hailin Jin, Shengjin Wang, Ming-Hsuan Yang, *Learning from PhotoShop Operation Videos: the PSOV Dataset*, In Asian Conference on Computer Vision (ACCV), 2018.

Han-Kai Hsu, Yi-Hsuan Tsai, Xue Mei, Kuan-Hui Lee, Naoki Nagasaka, Danil Prokhorov, Ming-Hsuan Yang, *Learning to Tell Brake and Turn Signals in Videos Using CNN-LSTM Structure*, In IEEE International Conference on Intelligent Transportation Systems (ITSC), 2017.

ABSTRACT OF THE THESIS

Learning to Detect in Images and Videos

by

Han-Kai Hsu

Master of Science in Electrical Engineering and Computer Science

University of California Merced, 2018

Professor Ming-Hsuan Yang, Chair

Learning to understand the visual context in images or videos is a challenging task in computer vision. Given an image, we can learn how to classify objects and further localize them by learning the spatial information of images. For videos, there are also tasks that aim to classify different sequences and predict their time interval within a video. In this case, not only does the spatial information matter, but we also need to exploit the temporal dependence of sequences in order to have a better understanding of the videos. In this thesis, we first learn to tell brake and turn signals of vehicles via a spatial-temporal model. Second, we construct a dataset and a deep model to explore how machines can help us in understanding the commands in Photoshop tutorial videos.

Many of these visual understanding approaches involve supervised learning models which rely on large annotated datasets. These models do not generalize well when the testing domain differs from the training one and it requires high labor cost to annotate another dataset. We address this issue in the thesis by proposing a progressive adaptation method to align the labeled domain with the testing one and conduct experiments on the object detection task. Our progressive adaptation relieves the difficulty under large domain discrepancy conditions by introducing an intermediate domain sitting in between the two domains and gradually adapt to our testing data distribution.

Chapter 1

Introduction

Enabling machines to understand the visual context of images and videos is a core problem in computer vision. For humans, we can look at an image or video and easily derive high-level understandings, i.e. object categories, locations, positions or relationship. However, it is challenging for machines to have similar visual comprehension as human. To this end, many research work has been proposed to obtain visual understandings via machines and tackle various problems. For images, we can identify existing objects and also localize them by drawing bounding boxes around them. Many object detectors are presented to approach this task, while the deep learning based detectors [63, 54, 65] achieve recent state-of-the-art detection results. Regarding videos, there are also tasks that aim to classify sequences and localize the corresponding time interval in the video such as action tube prediction [35]. In addition to learning the spatial information in images, video understanding requires to also exploit the temporal dependence of the image sequence. In order to extract temporal information from image sequences, various methods are presented to learn the frame to frame relationship, i.e. optical flow [59] and SIFT flow [53]. There also exist numerous deep learning methods for learning long-term dependencies such as the Long Short-term Memory (LSTM) model [37, 97] as well as the 3-D Convolution Neural Networks [44, 84]. In this thesis, we combine different approaches to extract the temporal features of sequences for better video understanding.

Many state-of-the-art approaches in the visual understanding tasks utilize supervised methods which rely heavily on data annotations. These methods often overfit on the

domain they are trained on and do not generalize well to different real-world scenarios. Due to the high labor cost in annotating additional data, unsupervised approaches are presented to address this problem, adapting knowledge learned from existing image labels and apply them to an unknown domain. To approach this issue, we focus on building an adaptive model and experiment on the object detection task.

In this thesis, we first aim to learn both spatial and temporal information from image sequences to achieve machine-assisted driving and understand software tutorial videos. Next, due to the generalization issue in supervised learning methods, we propose an unsupervised approach to align different data distributions in order to make use of existing resources and apply to other unseen domains.

In Chapter 2, we present a method that learns to tell vehicle rear signals from a number of frames using a deep learning framework. The proposed framework extracts spatial features with a convolution neural network (CNN) and then applies a long short-term memory (LSTM) network to learn the long-term dependencies. The brake signal classifier is trained using RGB frames, while the turn signal is recognized via a two-step localization approach. The two separate classifiers are learned to recognize the static brake signals and the dynamic turn signals. As a result, our recognition system can recognize 8 different rear signals via the combined two classifiers in real-world traffic scenes. Experimental results show that our method is able to obtain more accurate predictions than using only the CNN to classify rear signals with time sequence inputs.

In Chapter 3, we present the PhotoShop Operation Video (PSOV) dataset, a large-scale, densely annotated video database designed for the development of software intelligence and proposes a baseline method for command classification. The PSOV dataset consists of 564 densely-annotated videos for Photoshop operations, covering more than 500 commonly used commands in the Photoshop software. There are more than 74 hours of videos with 29,204 labeled commands. We believe that this dataset can help advance the development of intelligent software, and has extensive application aspects. To demonstrate that the PSOV dataset has sufficient data and labeling for data-driven methods, we develop a deep learning based algorithm for the command classification task using a 3-D convolution neural network. Due to the characteristic of software videos, we design an attention-aware preprocessing method to draw attention to operation-critical

regions. We also carry out experiments and analysis with the proposed method to encourage better understanding and usage of the PSOV dataset.

In Chapter 4, we propose a progressive adaptation method to approach the generalization issue caused by the discrepancies between domains. Our algorithm bridges the domain gap with an intermediate domain and then progressively solve easier adaptation subtasks. This intermediate domain is constructed by translating source images to mimic the ones in the target domain. To tackle the domain-shift problem, we adopt adversarial learning for aligning distributions at the feature-level. Additionally, a weighted task loss is applied due to the uneven image qualities in the synthetic domain. Experimental results show that our method performs favorably against the state-of-the-art method in terms of the model test performance on the target domain.

At last, we conclude the thesis in Chapter 5 and discuss possible future directions.

Chapter 2

Learning to Tell Brake and Turn Signals in Videos

2.1 Introduction

In recent years, autonomous driving has drawn significant attention, especially on the topic of safety. Human drivers communicate lane changes and turns via the car’s rear signal lights; thus, it is important that self-driving vehicles are taught to comprehend what each signal is, and when each signal is used. This is paramount in ensuring the safety of the passenger in either autonomous or assistive driving systems.

Numerous methods have been proposed to recognize brake and turn signals in the past decade. Existing systems mainly use hand-crafted features such as color thresholds or luminance to detect and extract the light regions [11, 55, 82, 51, 10]. As these manually defined features are variant due to different lighting conditions or clutter, existing methods based on such visual cues do not perform well in real-world scenes.

In this work, we identify rear light signals through learning deep features using CNN and LSTM networks. CNNs have been widely used in vision tasks and provide effective representations on learning the spatial features of images. On the other hand, the LSTM networks have been shown to be capable of handling the long-term dependencies in a sequence. Existing CNN-based methods for classification of brake signals [93] operate on the premise that brake signals are static and can be classified using only spatial

features. However, turn signals are dynamic and cannot be effectively recognized with one single frame, especially when the signal flashes on and off during that frame. Thus, we combine an LSTM network with a CNN to learn the temporal information of a sequence. Additionally, we develop a two-step localization approach to extract visual cues from the turn signals. First, we use the SIFT flow [53] of two continuous frames and warp the latter one based on the flow to align two frames. Then we compute the absolute difference between the warped image and the first frame to obtain the frame differences. Moreover, we focus on the tail light regions using region of interest (ROI) to crop out the input and send to the network as the additional guidance. As a result, our method is able to achieve higher classification accuracy than using RGB frames as the input.

Our training process uses a data set collected under real-world traffic conditions during the daytime. We define a total of 8 different rear signal states and label the data accordingly for supervised learning. Afterwards, we train classifiers for brake and turn signals individually and then integrate them into our rear signal recognition system. As a result, our proposed system can effectively identify 8 rear signal states, which outperforms the CNN only approach.

2.2 Related Work

For intelligent vehicles, it is imperative to distinguish between brake and turn signals, and learn when each is used. Numerous brake recognition methods [11, 55, 82] use thresholds on color features in order to detect the states of brake signals. Learning approaches have also been applied to learn feature representations for brake signals. Zhong et al. [98] train a FCN network [56] to identify the light regions and extract features within the region for classification using a linear support vector machine (SVM) classifier. In addition, deep convolution neural networks have been used to tell the state of brake signals after vehicle detection [93].

Turn signal recognition methods based on color thresholds have been developed [51, 10]. In these methods, turn signals are recognized by separating frames into left and right halves to determine the on and off states based on the number of pixels in the frame preserved by thresholds [51]. On the other hand, Chen et al. [10] train a

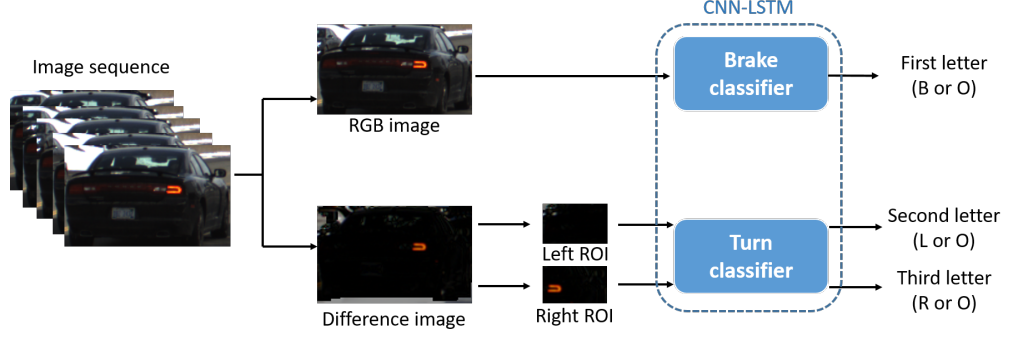


Figure 2.1: The brake and turn signal classifiers utilized in this system are based on a CNN-LSTM structure. The proposed system integrates the two classifiers with a 16-frame sequence as the input and outputs the rear light labels represented by three letters: B (brake), L (left) and R (right). We receive O (off) when the corresponding signal is turned off. RGB frames are passed to the brake classifier to obtain the first letter of the sequence state. Then, the computed difference image of the sequence is cropped into two ROI regions as input for the turn classifier and outputs the last two letters of the sequence state.

classifier using the AdaBoost algorithm to determine the existence of turn lights, and then use reflectance contrast to tell the directions. Alternatively, the method [26] locates the turn lights by using the Kanade-Lucas-Tomasi feature tracker [83] to compute the feature correspondence between two successive frames. Afterwards, the RANSAC algorithm [25] is used to receive the transformation matrix that maps the correspondence between the two frames. The latter frame is then transformed and subtracted from the first frame to get the absolute difference for locating the turn signal. Features are then extracted from the located regions and transformed to the frequency domain, in which an AdaBoost classifier is trained to identify the turn lights.

Many methods have been proposed for recognizing both brake and turn signals. Chen et al. [12] determine the lighted pixels by defining a response function and apply it to every pixel to locate the rear lights. A high-pass mask is used to find the illuminated region with brighter inner pixels for determining the states accordingly. Kalman filters and codebooks are used in [2, 3, 8] to track rear lights. The states are classified based on the luminance channel observed over time on the two detected light regions. Recently, a method that classifies rear lights into four states with an SVM [19] is developed where

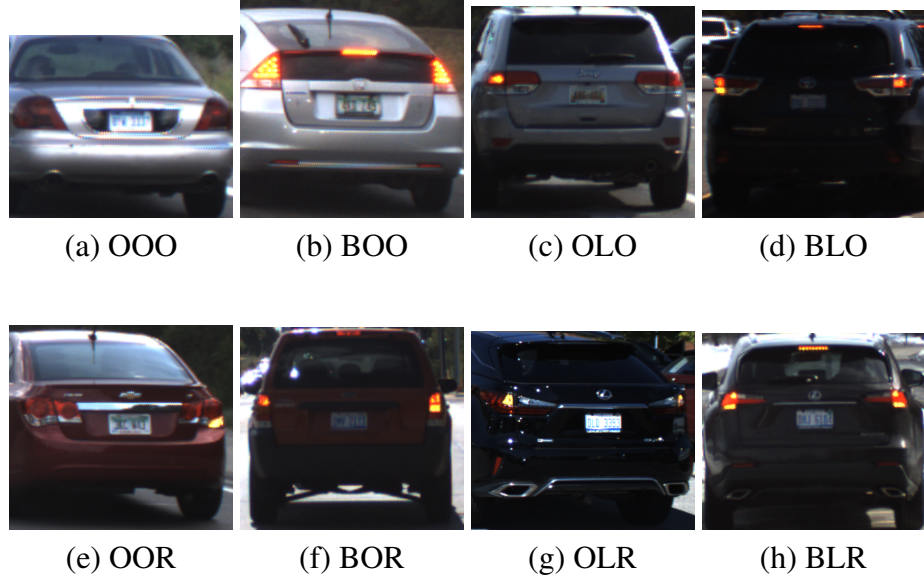


Figure 2.2: Examples of rear signal states.

the turn lights are detected based on thresholds.

2.3 Proposed Algorithm

In contrast to existing methods that are mainly based on hand-crafted features, we propose an algorithm to learn feature representations from the training data. Instead of only learning the spatial features through a CNN [93] to recognize brake lights, we also aim to recognize turn signals. Turn signals preserve different characteristics compared to brake signals. While flashing, it is difficult to recognize from one frame whether the turn signal is on or not at that time step. Thus, we use an LSTM module with a CNN to facilitate learning the temporal features of turn signals. We use a similar network as LRCN [21] which is developed for action recognition. This allows us to learn the different actions of rear signal states throughout a sequence. Our network takes a time-sequence as the input and determines the label accordingly. The CNN is used to extract spatial features from an input frame sequence, and the following LSTM module receives the output features from the CNN and learns the temporal information accordingly. Finally, we use the learned classifiers in our recognition system to determine the state of an input sequence in the real-world scenes. In the following section, we first describe

our recognition system including the brake and turn signal classifiers, and then proceed with the details of our CNN-LSTM framework used to train the classifiers. Fig. 2.1 shows the overview of the proposed algorithm.

2.3.1 Rear Signal Recognition System

Our recognition system has two main components including the static brake classification and the dynamic turn signal classification. As such, we can focus more on the specific characteristic of each task and achieve better performance. Toward this end, we learn two separate classifiers for brake and turn signals, then integrate them into our recognition system.

We define a total of 8 distinct states based on all combinations of brake and turn lights. Each state is denoted by 3 letters of B (brake), L (left), and R (right). We give either the corresponding letter of the signal when it is on, or a letter O for off. Consequently, there will be 8 different states as shown in Fig. 2.2. Accordingly, we annotate all frames in the training and testing sets using the state definition.

Brake classifier. RGB frames are used as input to the network for learning the brake classifier. We extract the spatial features through the CNN (see Section 2.3.2), and then exploit additional temporal features in a sequence.

Turn signal classifier. When training the turn signal classifier, we notice that extracting spatial and temporal information from RGB images is not sufficient for the system to recognize turn signals. We observe that there are different ways for vehicles to signal a turn. One type uses an individual light as the turn signal, and the other type utilizes the flashing brake lights when the turn signal is on. This makes it more difficult to distinguish between the brake and turn lights. In addition, the flashing turn signal occupies a relatively small area of the whole rear region and it is hard to extract a sufficient amount of visual cues from the turn lights. To resolve these problems, we propose a two-step turn signal localization method by first replacing RGB frames to the difference image of subsequent frames aligned using the SIFT flow [53]. As a result, there will be no differences on states besides turning and we can emphasize turn signals without being



Figure 2.3: Computing the difference between two consecutive frames. (a) Frame from previous step X_{i-1} . (b) Current frame X_i . (c) The computed SIFT flow [53] between X_{i-1} and X_i . (d) W_i is the warped image of X_i using SIFT flow in (c). (e) Absolute difference of W_i and X_{i-1} .

affected by other information like brake lights. In addition, we use ROI to exploit more information from small regions of turn lights in the frames.

Two-step turn signal localization. The first step is to find the frame difference. A difference image contrasts the change in turn signal while other parts of the image are subtracted and have no differences. However, we may obtain noisy results by directly computing the absolute difference because the vehicles in the two frames are not aligned well. To obtain better difference images, we use the SIFT flow algorithm [53] to help align the vehicles in successive frames. It has been shown that images in the same scene can be effectively aligned using the SIFT flow [53]. As the input sequences considered in this work are from the same scene, the SIFT flow is likely to help obtain better difference images. We take two adjacent frames in the video sequence X_{i-1} and X_i , and compute the SIFT flow. Next, we warp X_i to W_i based on their SIFT flow to align two images. We then obtain the difference image by computing $|W_i - X_{i-1}|$. An example of a difference image based on the above operations is shown in Fig. 2.3(e).

In the next step, we use ROI to focus on the rear light regions for better feature

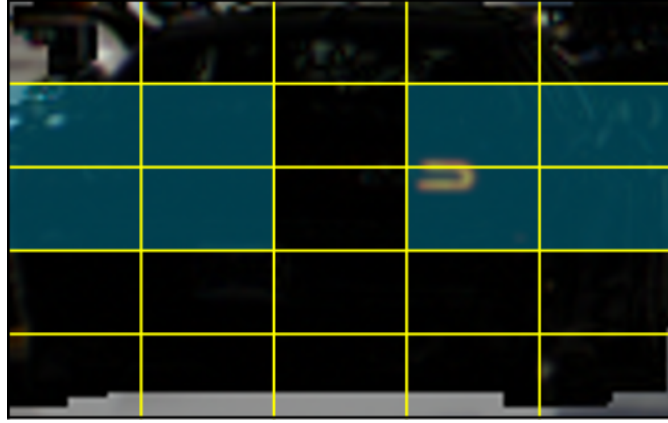


Figure 2.4: ROI is indicated as the blue highlighted area, where the left and right blue regions are propagated through the network separately.

extraction. Our ROI is represented as the blue highlighted region in Fig. 2.4. During the learning process, we take the two regions of our ROI from the computed difference image as separate inputs to our network. Consequently, the network learns to distinguish whether that ROI region is flashing or not.

Our recognition system utilizes two classifiers to successfully recognize 8 different signal states. First of all, we pass the RGB frames directly to the brake classifier and determine the state of the brake signal. Second, we compute difference images from the given sequence and extract the left and right ROIs as two inputs. These two light regions then propagate through our turn classifier to determine the states of the left and right signals respectively. The states (represented by three letters) from both classifiers form the prediction for an image sequence.

2.3.2 Convolution Neural Network

The CNN is able to learn the spatial features of images and has been successfully applied to many different image classification challenges. Therefore, we apply the CNN to learn the underlying features of our input. The CNN serves to exploit the spatial features within every frame in order to further learn the long-term dependencies in the sequences.

Our network adopts the CNN-LSTM structure as shown in Fig. 2.5. The CNN

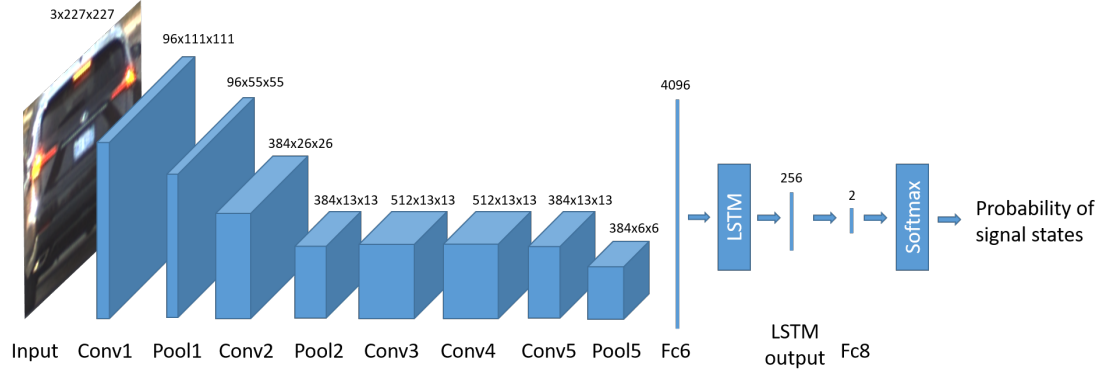


Figure 2.5: Illustration of our network architecture. The layers before fc6 serve to extract spatial information from our input. Afterwards, we send the features to the LSTM module to utilize the temporal information and further classify the desired state in the last fully-connected layer.

network for the spatial features extraction is a variant based on CaffeNet [45], which is also utilized in [21]. There are nine layers in total, including five convolutional layers, three pooling layers and a fully-connected layer. We utilize this CNN to learn the tail light features of the vehicles for localization and classification. After obtaining these spatial features, we feed the information of each frame to the LSTM module in order to learn the temporal information in each image sequence.

2.3.3 Long Short-Term Memory

The LSTM model, derived from the the recurrent neural network (RNN), has been used in various tasks (e.g. speech recognition, action recognition) to learn the temporal information of the sequence inputs. The advantage of the LSTM over the RNN is that it effectively alleviate the vanishing gradient problem in the RNN. The LSTM unit incorporates several non-linear activation gates to determine whether to retain or discard the information. There are numerous variants since the first LSTM model [37]. In this work, we use the LSTM unit proposed in [97].

The LSTM module is used in our network to maintain the long-term dependencies of our input sequences. Since obtaining the spatial features from the CNN is not sufficient to classify turn signals, it is essential for the classifier to learn the temporal information

throughout the sequences to determine the dynamic flashing states.

The structure of the LSTM model in this work is shown in Fig. 2.6. One input is the feature x_t from the fc6 layer of the CNN described in Section 2.3.2 at time step t , and another input is the hidden unit h_{t-1} from the last time step. At every time step, the LSTM unit estimates the hidden unit h_t and sends it to the LSTM unit at the next time step. The LSTM unit also receives a memory cell c_{t-1} which holds the information from previous time steps. The memory cell is updated at every time step and then passed to the next LSTM unit. All the updates and outputs are processed by the different gates of the LSTM unit. First, the forget gate f_t determines what to discard from x_t and h_{t-1} . A forget gate is a sigmoid function (σ) that outputs values from 0 to 1 and performs element-wise product ($\odot$) with previous memory cell state c_{t-1} to determine what information to forget or remember. Next, the LSTM unit updates information to the memory cell through the input gate i_t and the hyperbolic tangent ($\tanh$) layer g_t . These two gates, i_t and g_t , control what information to remember from x_t and h_{t-1} then add them to the memory cell. The activation gates are computed by

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (2.1)$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (2.2)$$

$$g_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (2.3)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (2.4)$$

where W denotes the weight connecting the inputs and the given gate, and b is the corresponding bias term.

Aside from updating the memory cell, the LSTM unit also outputs a hidden state h_t at every time step. The output gate o_t is computed and weighted with the cell states through a $\tanh$ layer to determine the hidden state h_t by

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (2.5)$$

$$h_t = o_t \odot \tanh(c_t) \quad (2.6)$$

The output of each LSTM layer is sent to the last fully connected layer of our network to compute a class probability for each time step. In order to take the temporal

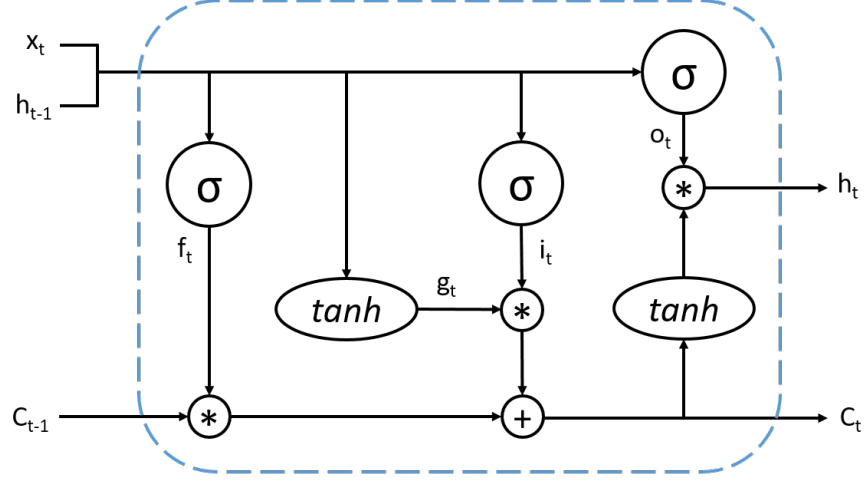


Figure 2.6: The LSTM unit in the proposed algorithm.

dependence of an input sequence into account, we focus on the output prediction of the last frame, which contains sufficient information from all the previous frames. That is, we compute the loss of the last frame and backpropagate all frames in the sequence with the same loss. Given a test frame, instead of taking average among the output predictions, we take the prediction of the last frame as the label for the entire input sequence.

2.4 Experimental results

In this work, we collect a data set of 649 videos including 63,637 frames. The sequences are recorded during the daytime under real-world driving conditions with various vehicle types. We crop out the image regions of car rears in each video and label them based on our definitions described in Section 2.3.1. It is difficult to collect videos for emergency lights (OLR and BLR), since such cases are rarely observed in daily driving conditions. However, our network is able to handle this problem without having many samples of these two particular cases. The proposed algorithm separates each image into right and left signal light regions. With sufficient amount of images in all the other turn signal states (OOR, BOR, OLO and BLO), we are able to tell whether the turn signals are on for both regions. Table 2.1 summarizes the properties of the collected dataset.

Table 2.1: Data set for turn and brake lights.

	OOO	BOO	OLO	BLO	OOR	BOR	OLR	BLR	Total
Videos	188	211	78	63	58	33	9	9	649
Frames	21,867	17,874	6,271	6,380	4,728	3,527	1,600	1,390	63,637

Table 2.2: Brake recognition accuracy (%) comparison with using the CNN and our method (CNN-LSTM). The results are based on five-fold cross validation.

	Brake ON	Brake OFF
CNN	94.52 ± 0.19	95.60 ± 0.03
Ours (CNN-LSTM)	94.80 ± 0.21	96.22 ± 0.03

2.4.1 Data pre-processing

Before training the network, we process the raw data such that we can have a sufficient amount of training samples for training the proposed CNN-LSTM network. In this work, we augment the raw data with random cropping, horizontal flipping, gamma correction, rotation, and shifting such that we increase the training samples by 300 folds.

For turn recognition, we compute the difference image of all the video frames. Additionally, we crop the ROI on the difference image, which is $2/5$ height and $2/5$ width of the right and left regions as shown in Fig. 2.4. Both regions are fed to the network as inputs and the training data is increased by two folds. We evaluate the proposed algorithm by five-fold cross validation.

2.4.2 Network settings

The proposed CNN-LSTM model is implemented using the Caffe toolbox [45]. We fine-tune the pre-trained model [21], which is developed based on the UCF101 data set [76]. The input of our network contains 10 batches each time, and each one contains 16 frames. A sequence of 16 frames allows us to obtain at least one cycle of the turn signal. Each frame is given the ground truth label based on the video sequence they belong to and resized to 227×227 pixels before feeding to the network. During the

Table 2.3: Turn recognition component analysis of the proposed algorithm with different combinations of features and modules. RGB/difference image (Diff), ROI, and LSTM are added for evaluation based on prediction accuracy (%).

	ROI	LSTM	No turn	Left turn	Right turn	Emergency
RGB	✓		88.90	59.82	43.63	0.00
Diff	✓		88.45	89.79	64.10	100.00
Diff	✓		90.85	80.50	64.47	100.00
Diff	✓	✓	97.94	94.57	87.72	100.00

Table 2.4: Turn recognition comparison with using the CNN and our method (CNN-LSTM). Both methods use difference image and ROI for inputs, and evaluated using five-fold cross validation.

	No turn	Left turn	Right turn	Emergency
CNN	91.32 ± 0.17	66.42 ± 0.82	64.46 ± 1.19	91.61 ± 1.17
Ours	95.92 ± 0.09	91.12 ± 0.23	93.46 ± 0.22	99.16 ± 0.04

error backpropagation process, we take the ground truth and the output prediction of the last frame to compute the loss for every frame in the same sequence.

As we have an unbalanced data set for the brake and turn signals as shown in Table 2.1, we randomly select the same amount of videos for the on and off of brake or turn signals in our 10 batches each time. As a result, we have sufficient amount of training samples for all classes.

2.4.3 Performance evaluation

For each test video, we select every unique sample of 16 frames sequence and feed it to the trained model (e.g., 2 sequences for a video of 17 frames). We compute the accuracy based on the percentage of correctly predicted sequences in each video and use the average as the metric for performance evaluation.

Table 2.2 shows the brake recognition results using the proposed algorithm and the CNN only method, which is implemented based on the AlexNet in the Caffe toolbox.

Table 2.5: Final recognition accuracy (%) evaluated using five-fold cross validation.

Category	Accuracy
OOO	93.52 ± 0.25
BOO	90.30 ± 0.38
OLO	86.12 ± 0.89
BLO	88.02 ± 0.44
OOR	94.72 ± 0.20
BOR	84.00 ± 1.23
OLR	97.90 ± 0.18
BLR	100.00

The CNN method is evaluated using the average accuracy from the predictions of all frames in each video. The results show that although brake lights are static and depend less on temporal features, our method is able to achieve better accuracy with additional temporal information.

We evaluate the turn signal classification results to analyze how the proposed model performs. Table 2.3 shows the experimental results with different features and modules including RGB/difference image, ROI, and LSTM. Experimental results show that we can increase the recognition accuracy of turn signals using all the modules and features in the proposed algorithm. In Table 2.4, we show the results using five-fold cross validation on the proposed algorithm in comparison to the method only using the CNN. Both networks use difference image and ROI as inputs for classification. The results show the effectiveness of the proposed algorithm using the LSTM model.

Table 2.5 shows the five-fold cross validation results for 8 turn signal recognition using the proposed algorithm. Overall, the proposed algorithm is able to tell the turn light signals effectively.

In the experiments, we observe some failure cases as shown in Fig. 2.7. For brakes, there are cases that are difficult to recognize due to the sunlight. For example, Fig. 2.7(a) is labeled as OOO but it looks like the brake signal is turned on. On the other hand, Fig.



(a) Brake failure (OOO) (b) Brake failure (BOO)



(c) 4 continuous frames of the turn failure case (OOO)

Figure 2.7: Brake and turn failure cases. (a) and (b) are brake recognition failures affected by strong sunlight and is difficult for human to recognize the signal state. (c) is a turn failure case caused by sunlight and shades which in result acts similar to flashing signals.

2.7(b) is labeled as BOO but it is difficult for humans to tell whether it is the color of the light mask or it is actually turned on. Turn signals have similar issues with additional factors such as shades. In Fig. 2.7(c), tail lights of the middle two frames appear to be on due to the sunlight, and the lights in the other frames have cast shadows. As a result, the image differences between the first two frames (and last two frames) appear to be similar to flashing signals. In such cases, the proposed CNN-LSTM model does not perform well.

2.5 Summary

In this work, we present a method that learns to recognize 8 different signal states. Our method utilizes a deep network to learn the temporal information through an LSTM

network in addition to exploiting the spatial features from the CNN. We train the brake signal classifier using RGB frames and use a two-step localization method to learn the dynamic patterns of the turn signals. First, we compute the difference between two adjacent frames. Second, we segment out the ROI for the rear light regions in the image as our input to the network. The two classifiers are learned separately and is then integrated to our recognition system. We show that our proposed system is able to effectively predict each signal states in the real-world traffic scenes.

Chapter 3

Learning from Photoshop Operation Videos

3.1 Introduction

Recent years have witnessed rapid development in software intelligence. With the performance leap made by deep learning, there is an explosion of works in automatic human-assisting techniques, e.g. advanced driver assistance system [9, 13, 50, 70], machine translation [4, 17], interactive robots [67, 81, 49], and virtual player [92, 72, 94, 31, 32]. As many state-of-the-art algorithms are data-driven, well-designed datasets [23, 1, 71] contribute a lot to this prosperity. For example, [24, 52, 20] boost the development in image classification, [78, 29] enable rapid progress in robotic vision; [46, 68, 77] assist researches in action recognition largely. In spite of the numerous existing datasets, there is still a lack of data for one particular use: computer software intelligence.

Computer software plays an important role in everyday life. Due to the never-satiable appetite of computer users, there exist rapidly-growing number of computer software, varying a lot in function, operation, and etc. Therefore, it is important for software to provide easy access for beginners. A common solution is to put all technical details in a user-guide. However, the long, boring, rich-text user-guide itself causes trouble for starters. We propose that software intelligence could be incorporated here to help solve this problem. Starters of a software can first refer to an intelligent agent,

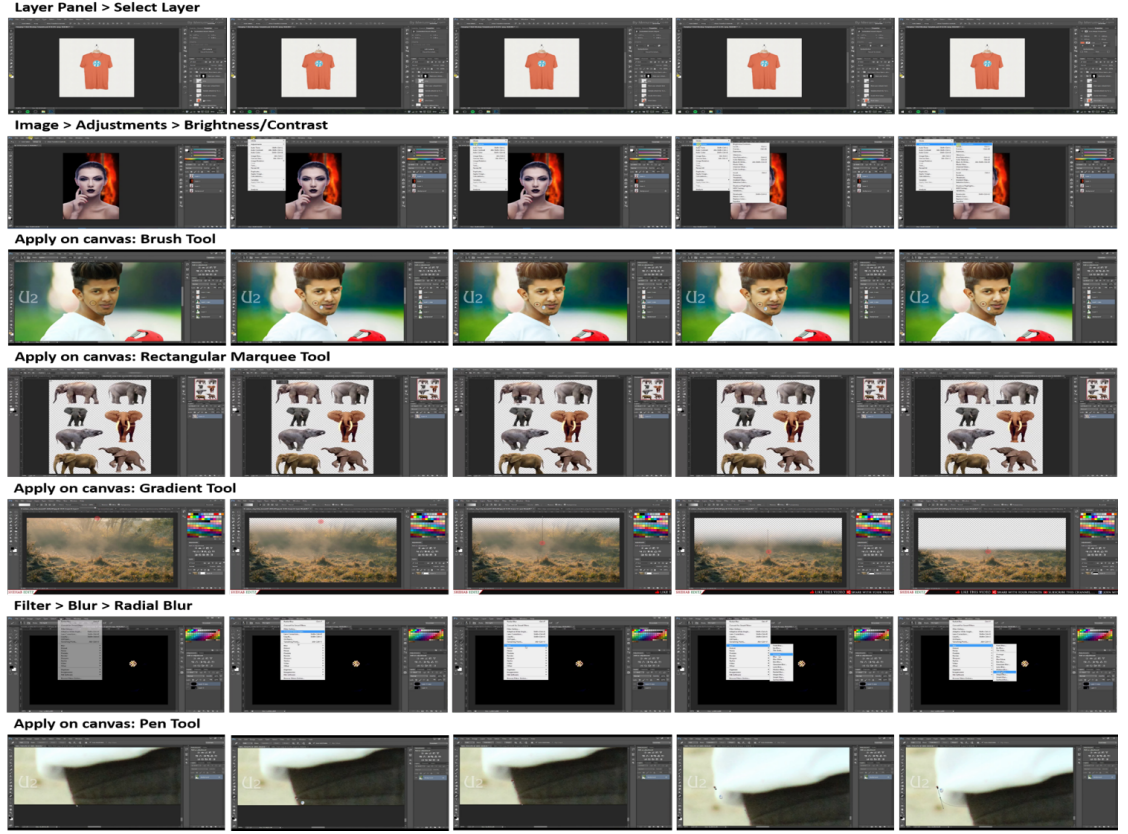


Figure 3.1: Example frames of the PSOV dataset. Each row represents a video clip for one specific Photoshop operation.

which briefly narrates shared Internet instructional videos, and advises users with instructional videos in correspondence with specific needs. With the help of these highly related, readily comprehensible instruction videos recommended by intelligent software agents, the software can be much easier to understand, operate and spread.

We consider software intelligence as a next research hotspot due to the predictable huge potentials. However, there are few published datasets designed to help algorithms understand software operations. The most closely related dataset, MiniWoB [71], aims to provide simulated environment and data helping software agents to learn interactive tasks on the web. But this dataset only uses synthetic video data, which has small window size (160×210 pixels), simple operations and primitive interfaces. Motivated by this observation, we propose to construct a computer software dataset that can further

encourage research and development of intelligent software in real-world situations. For concreteness, we first focus on one widely-used software with hundreds of complex operations, Photoshop. We collect a large number of Photoshop operation videos (mostly instructional videos), annotate them and also propose some tasks for easy entry.

In this work, we present a large-scale, densely-annotated PhotoShop Operation Video (PSOV) dataset (Figure 3.1). The PSOV dataset contains videos and dense command annotations for real-world Photoshop Software. Each annotation includes command name, start and end time accurate to seconds. There are 74 hours of videos and 29,204 labeled commands in the dataset. In addition, we define three tasks on the PSOV dataset: command classification, command tube prediction, and command recognition. The details of each task and evaluation metrics are described in Section 3.3. To have a better insight of the proposed dataset, we also construct a 3-D convolutional neural network based algorithm for the command classification task. By experimenting with the proposed network, we validate that the PSOV dataset is capable of supporting deep learning methods, and encourage further understanding of this database. The dataset, task definition, evaluation code as well as annotation tool are available at <http://vllab1.ucmerced.edu/hhsu22/PSOV/>.

The main contributions of this work are: 1) a first-of-its-kind, large-scale, real-world PSOV dataset containing dense command annotations; 2) three well-designed tasks with evaluation metrics to help develop software intelligence; 3) a baseline algorithm for better usage and comprehension of the proposed dataset.

3.2 Dataset Construction Procedure

Raw videos for Photoshop operations are downloaded from YouTube¹. The videos are collected using the Youtube Data API², which allows users to search for corresponding video information (such as video title, views, likes, and duration) using keywords. We use keywords like *Photoshop*, *Photoshop Introduction*, *Photoshop Operation* and *Photoshop Tutorial* to search for potential Photoshop videos. The API does not return all the related videos on Youtube due to some restrictions. In order to look for as many videos

¹www.youtube.com

²developers.google.com/youtube/v3/

related to the given keyword as possible, we set different time windows and make multiple searches for each keyword. We take the union of all the search results and remove duplicate videos programmatically. We also filter these videos with the requirement of a minimum 720p resolution. This procedure results in a collection of 184,626 videos. We observe that videos which are more related to Photoshop operations often have some creator input, i.e. caption data in the video metadata file. To guarantee high quality, we only keep the videos which have caption data, resulting in 3,734 remaining videos. Then, we go through captions of each video and sort out more than 2,000 low-quality or non-related ones. Finally, each video is watched and evaluated manually until we reach the final 564 high-quality Photoshop operation videos.

For labeling, we use a crowdsourcing platform due to the huge amount of this work. We annotate each and every command performed in the collected videos with the help of several workers, who has experience in using Photoshop. The workers are hired from Upwork³, a global freelancing platform which enables remote communication and collaboration. Upwork provides the option to specify skill-level requirements for tasks, allowing us to hire workers with a certain level of Photoshop software knowledge. Other than their Photoshop skills, we also set a rating requirement for workers (each worker has an averaged rating from their previous jobs). Only the top-ranked workers in the platform are invited for our labeling project. Before labeling, we also conduct an interview and a training process to ensure the qualification of the workers.

During the annotation process, we implement an online annotation tool to help facilitate remote working as well as simplify the labeling process (released with the dataset). Figure 3.2 shows an interface of the annotation tool, a web application based on the Express.js framework⁴. The tool shows a progress bar for each video, enabling workers to easily navigate through a video and precisely locate commands. In the bottom-right corner, there are two additional time bars designed for fine adjustment of the start and end time point, respectively. These progress bars each represents a 20-second interval with the selected time point in the center. They contribute a lot to time precision during annotation. Workers can select an approximate time on the full-length progress bar, and make small adjustments here to be accurate to seconds. Users can change the

³www.upwork.com

⁴www.expressjs.com

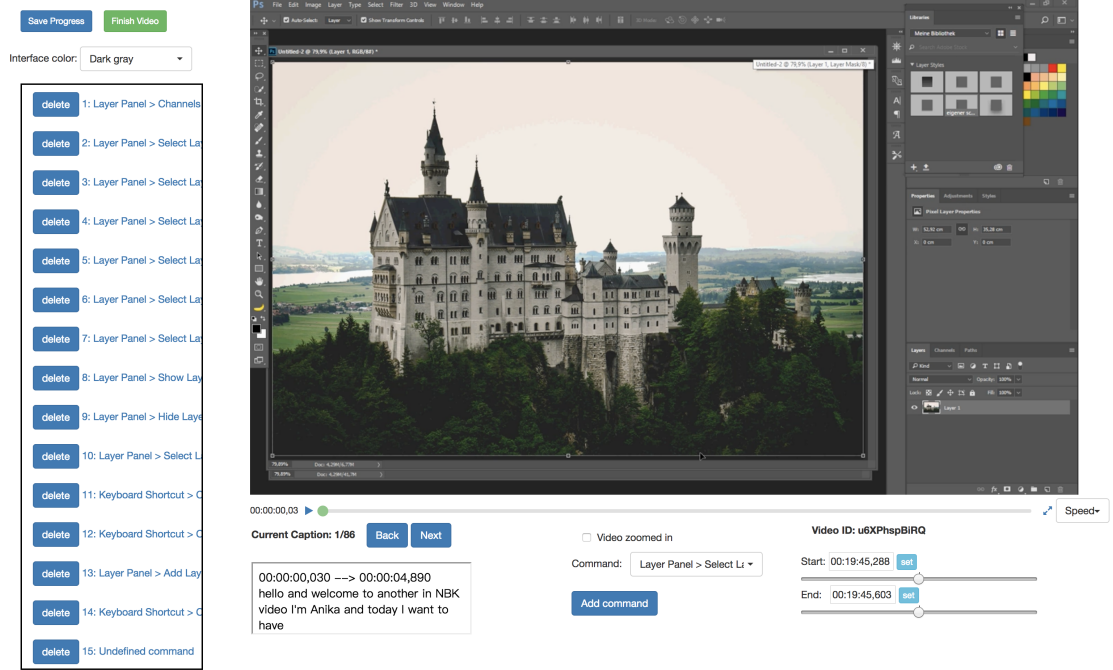


Figure 3.2: Labeling tool for Photoshop videos. This figure shows an example interface of our online labeling tool, by which workers can easily annotate a Photoshop operation video. Past annotations are listed on the left. See detailed descriptions in Section 3.2.

video playback speed (next to the full-length progress bar) in case that the candidate video is fast forwarded by its creator. We also consider other factors like interface color which differs due to software versions or themes, and video zoom-in selection when the software does not occupy the full screen (e.g. the bottom row in Figure 3.6).

Before starting the labeling process, each worker is assigned an account name for user identification. Workers need to log in to their own account to start labeling. Videos are assigned randomly to workers one at a time, with the text input from video creator as a reference during labeling (see the text box in the bottom-left of Figure 3.2). Workers are allowed to add, delete, and insert command labels. The labeling process also requires the user to select an interface color of the Software. In the labeling process, we set these two pre-defined colors for workers to choose from (dark gray and light gray). The most important and time-consuming part is for workers is to label the start and finish time, as well as the specific operation name of each command. Along with that, workers

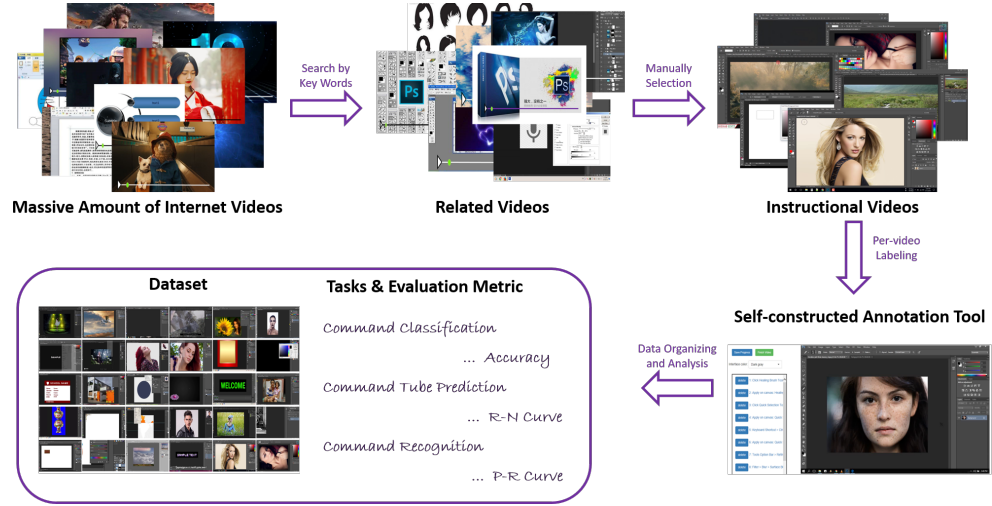


Figure 3.3: Pipeline of dataset construction procedure.

are required to judge whether the entire software interface is within the screen during each command, because videos may be post-edited by creators to zoom into a specific region in some cases, and video frames vary a lot after zooming in (see bottom row in Figure 3.6 for an example). After labeling the candidate video, workers can either click *Finish Video* to receive the next assignment and upload the current one onto the server, or click *Save Progress* and return to where they left off afterwards. Finally, the labeled commands are double checked by ourselves to ensure the correctness.

We show the pipeline of our dataset construction procedure in Figure 3.3, and describe details of the PSOV dataset in Section 3.3.

3.3 Dataset Description.

In this Section, we introduce the PSOV dataset, a large-scale, densely-annotated video dataset, specially designed for development of software intelligence. Example frames of some typical Photoshop commands are shown in Figure 3.1; and the dataset structure is illustrated in Figure 3.5.

Data Amount and Quality. The PSOV dataset consists of 564 densely-annotated Photoshop operation videos. There are 74 hours of video with 29,204 labeled commands. Each video has the minimum resolution of 720p. Labels of command operations in Pho-

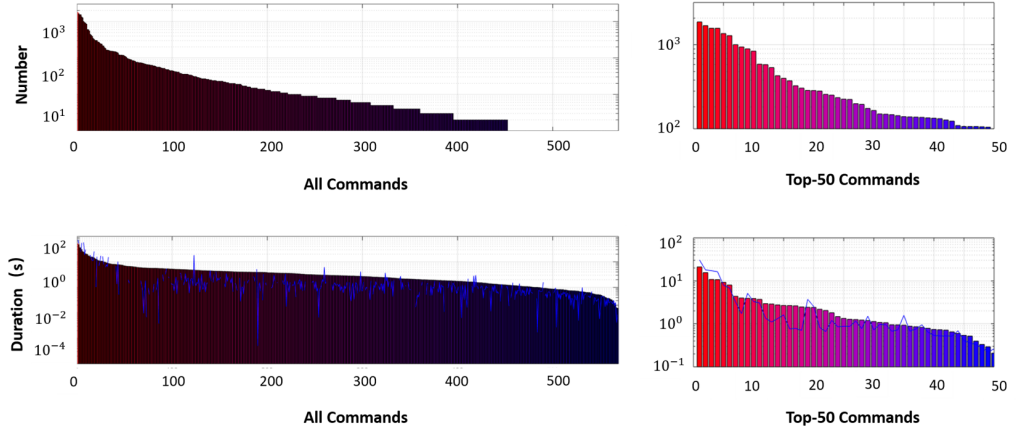


Figure 3.4: Data distribution in the PSOV dataset. From left to right, we show data distribution of all labeled commands and the top-50. The first row shows the number of each command, and the second row shows the average duration (finish time minus start time) of the commands, all sorted by value. The blue lines in duration figures denote duration variance values of each bar. The top-left figure shows that about 150 commands only have 1 labeled sequence, and that the top-50 commands have the number of sequences more than 100. We provide labeling data for all command sequences in the dataset, but only evaluate tasks on the top-50 to guarantee enough training data.

to shop are predefined by ourselves by exploiting user guides, technique books, etc. The command definition is in a concise and effective manner, for instance, *Layer Panel > Select Layer, Image > Adjustments > Brightness/Contrast*, and *Apply on canvas: Brush Tool*. All 29,204 commands are labeled by the workers hired from Upwork (see details in Section 3.2). Note that besides a large portion of usual mouse click interactions, the labels also include keyboard short-cuts (e.g. Control-N, Control-C, Control-P) which are often used in Photoshop software. These keyboard short-cuts make the dataset more challenging and more realistic, since they are hard to recognize for beginners. The number of samples for each command is shown in Figure 3.4. We select the top-50 commands (those with larger amount) for tasks on this dataset to ensure a sufficient volume of data for data-driven techniques like CNN-based video processing algorithms [73, 61, 47, 96, 44]. In Section 3.6, we demonstrate that the PSOV dataset has enough

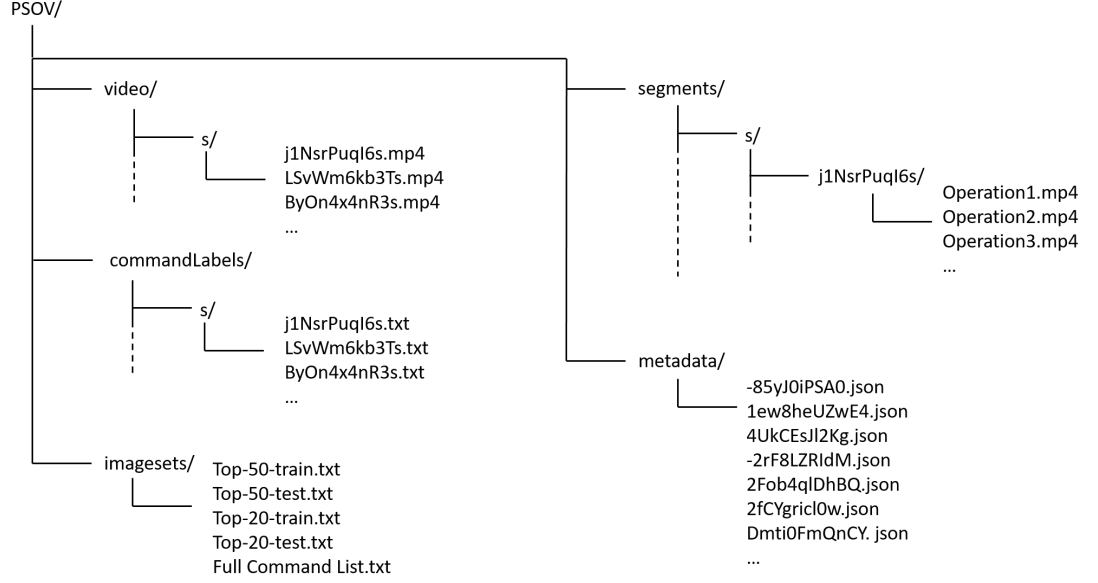


Figure 3.5: Dataset structure. This figure illustrates the data storage path in the dataset, where folder *video*, *commandLabels*, *segments*, *metadata* contains whole videos, per-video annotation file, operation frames, and video caption information respectively.

data quantity and diversity for training deep-learning based algorithms.

Challenges. The PSOV dataset is a challenging dataset, as the video data collected have fast but minor motion, with large variance in duration and background (see Figure 3.6 for an example). In addition, data samples of different commands are imbalanced. Figure 3.4 presents an example of the duration difference between two command sequences with the same label *File > New*, showing a key difficulty of time variance in this dataset. The figure also gives examples of two other challenges: minor motion and background clutter. The PSOV dataset holds many sequences where motion happens in a tiny local area that can not be distinguished by current optical flow methods [80, 66, 22, 42]. Furthermore, some operation sequence may contain severe background clutter (zooming-in, panel change, etc.), causing confusion for recognition. We analyze the influence of some challenging factors using the proposed method in Section 3.6, providing a better insight and understanding of the PSOV dataset.

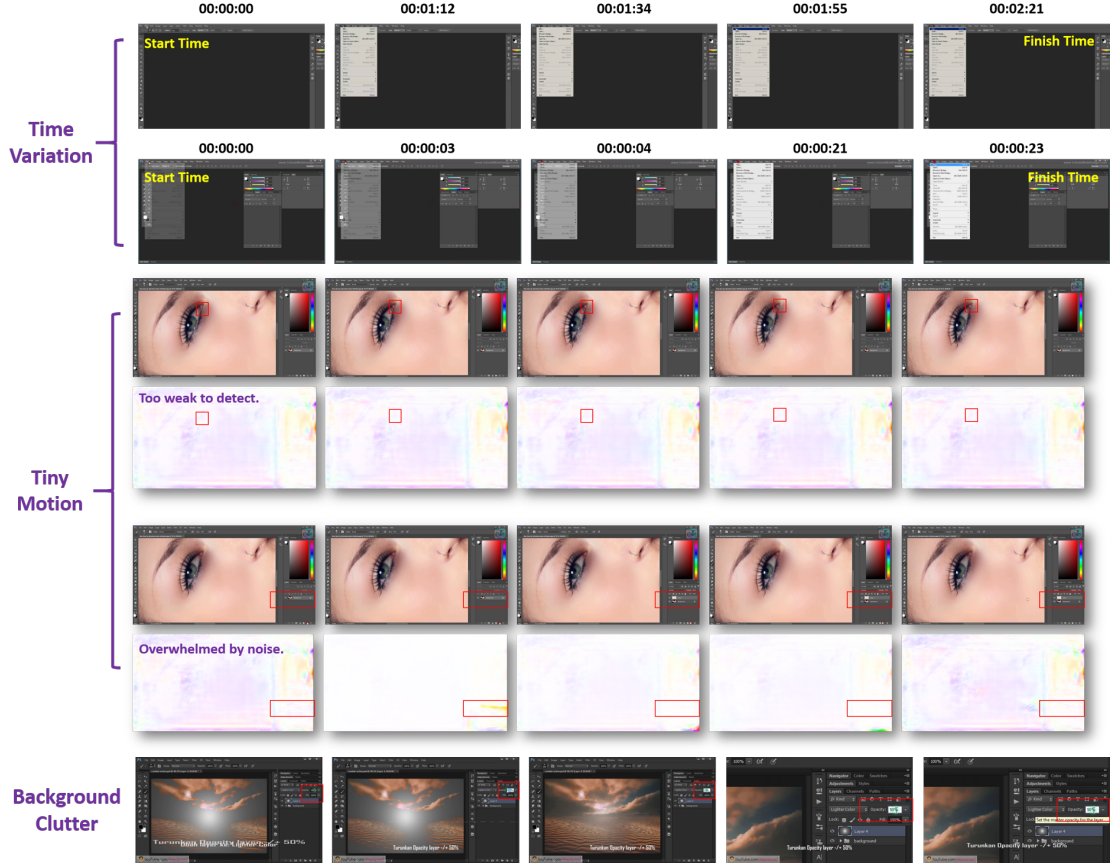


Figure 3.6: Key difficulties in the PSOV dataset. This figure shows examples of three key difficulties in the PSOV dataset, i.e. duration variance, tiny motion and background clutter. 1) For time variance, each row shows a sequence of class *File* $>$ *New*. This same operation has a two-second difference in the two sequences. 2) For tiny movement, optical flow [22] is used to illustrate what can be seen in motion space: the brush moving around lower eyelid in the first sequence is too weak to be detected; and the new layer appeared in the right panel has no stronger response than background noises. 3) For background clutter, we show an extreme example where the center region is zoomed in by the creator, largely changing the background of the operation.

3.4 Tasks and Evaluation

In this section, we describe three tasks as well as the corresponding evaluation metrics on the PSOV dataset. Tasks share common training and testing set, which contains

433 and 131 videos, respectively (training and testing sets are split in a manner that the command distributions are similar). To ensure a sufficient amount of labeling data for deep learning methods, we only conduct these tasks on the top-50 and top-20 popular commands. We also provide the evaluation functions in the development kit to release together with the dataset.

Command Classification. The command classification task aims to recognize the operation performed in a given video tube⁵. In this task, the start and finish time of commands in both training and testing sets are given. Algorithms need to learn a classifier from the 433 training set videos and predict the command label for the operation tubes in the test set. We use the simple and intuitive classification accuracy to evaluate the performance of different methods on this task. Using our development kit, the classification accuracy and per-class precision will be given once obtaining a 50-dimension probability vector from the algorithm.

Command Tube Prediction. This task aims to predict the begin and finish time of each command in Photoshop videos. With the available training set videos and corresponding command labels, methods need to predict the two time points (start time t_{start} and finish time t_{end}) for each operation in the test set videos. We propose to use a R&N Curve⁶ as the evaluation metric, where a tube is considered 'hit' when a proposal has the IoU (intersection over union between ground truth time interval and the proposal) greater than 0.5. Note that commands not in the top-50 are not calculated in this task. The methods on this task predict proposals for command tubes ($[t_{start}, t_{end}]$); and are evaluated by R&N Curve where higher curves denote better performance.

Command Recognition. Command recognition is a comprehensive and the most complicated one among three tasks. This task is a further step from classification and tube prediction, it aims to recognize commands (predict start time, end time, command name of operations) from a raw video. Given a test video, the algorithm needs to decide which time period exists an operation and exactly which one it is. It is closest to reality, as the method understands when and what commands are performed in videos with no manually provided information. Algorithms for this task can be directly applied to

⁵Video tube denotes a sequence of video frames which contains one specific command.

⁶R denotes recall, and N denotes the number of proposals averaged over the number of ground truth commands.

Photoshop operation videos outside of the PSOV dataset. They can sketch instructional videos with step by step operation list, relieving users from browsing over tens of thousands of video searching results. They can also dig useful data from massive amount of videos uploaded to the Internet every day, and provide assistance to researchers and software developers. Furthermore, their output operations can also be transformed back into computer commands, so that the computer can reproduce automatically in real-world software. The command recognition task is evaluated by AUC (the Area Under precision-recall Curve). Note that the correct prediction here has an IoU with the ground truth over a certain threshold and a correct command label prediction. Both precision-recall curve and AUC value are provided by development kit in evaluation.

3.5 Methodology

We develop a command classification algorithm on the PSOV dataset to: 1) show an example usage of the proposed database; 2) validate that the PSOV dataset has sufficient data volume for developing data-driven algorithms; and 3) provide a baseline comparison for the command classification task. This section describes the details of our method construction.

Convolutional neural network (CNN) plays an important role in computer vision these years for the effectiveness and robustness of CNN features and classifiers. Many algorithms [96, 15, 7, 33] use CNN for video recognition, which usually process each frame independently and use feature fusion to obtain video descriptions. However, such methods make little use of the motion information in time dimension since the feature of each frame is extracted separately. In this work, we propose to use a 3-D CNN [44, 84] for the challenging PSOV dataset. First, we design an attention-aware preprocessing method to draw attention to operation-critical regions. Then we regularize each video to a fixed length with reference to the attention information. Finally, a 3-D CNN structure is trained for the command classification task.

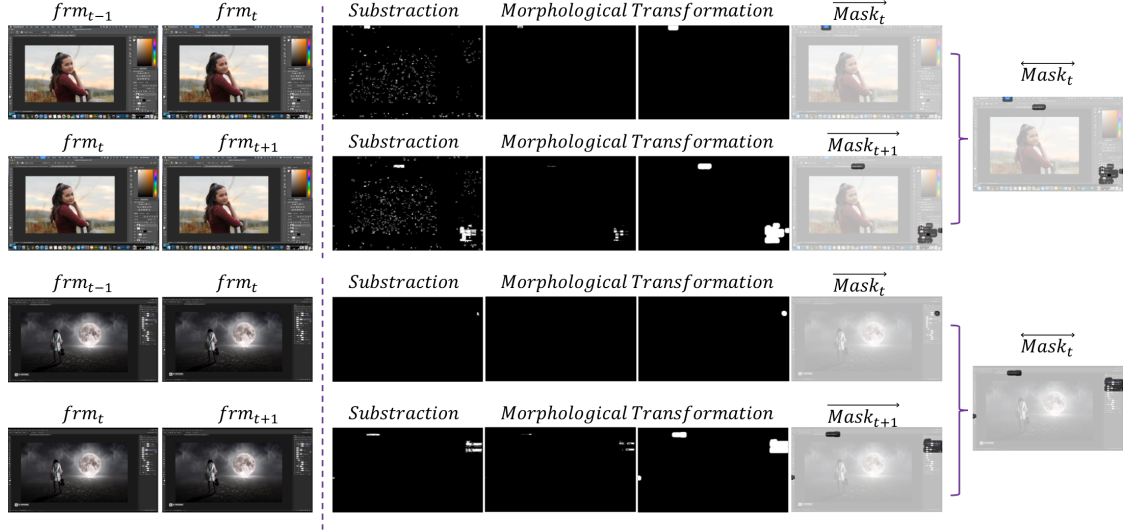


Figure 3.7: Attention-aware filtering. We show the attention filtering results on example frames from two commands: *Layer Panel > Select Layer* and *Layer Panel > Duplicate Layer*. $\overrightarrow{Mask}$ denotes the attention computing in single direction ($frm_{t-1} \Rightarrow frm_t$, $frm_t \Rightarrow frm_{t+1}$); and $\overleftarrow{Mask}$ denotes the result considering bi-directional context ($frm_{t-1} \Rightarrow frm_t \Leftarrow frm_{t+1}$).

3.5.1 Attention-aware Filtering

In video-related tasks, it is common to leverage temporal features [16, 75]. Optical flow is one of the most commonly used descriptors for such information. However, this traditional feature does not take effect in the PSOV dataset, for that key motions are usually weak or located in a tiny region in this dataset (Figure 3.6). If we use the whole image frame as network input, information from key regions can easily be overwhelmed by surrounding background noises. Thus, we propose an Attention-aware Filtering algorithm that directly extracts features from the strongest motion part by filtering out useless area. Figure 3.7 shows an example of the process for our filtering method, which helps the network to focus more on the informative and effective region, boosting its recognition ability (see Section 3.6.1).

Difference Filtering. The purpose of our Attention-aware Filtering method is to focus on the informative motion region. As shown in Figure 3.6, the motion that determines specific operation often takes place in a small fraction of area. An intuitive way to find

this area is to use the difference map between two adjacent frames ($frm_t - frm_{t-1}$). However, due to video compression artifacts, the direct subtraction has a noisy result (column three in Figure 3.6), making it difficult to locate true movement. To deal with this phenomenon, we propose to use morphological image processing methods: erosion and dilation. First, we apply erosion with a disk-shaped kernel (radius 1 pixel) on the subtraction result, removing noisy points here and there. As the erosion procedure comes with region shrinking, we then apply a dilation kernel (this time by a disk-shaped kernel with a radius of 20 pixels) to ensure that most information remains in the outcoming mask ($\overrightarrow{Mask_t}$). Figure 3.7, shows that Difference Filtering can effectively locate main movement region, relieving the difficulty caused by minor motion in the PSOV dataset.

Bi-direction Context. As movement happens between two frames, context information is needed for both before and after the action ($frm_{t-1} - frm_{t+1}$). While $\overrightarrow{Mask_t}$ is calculated between two frames, it only knows what happened before the action but has no idea about the temporal context afterward. This can cause serious information loss. For example, command *Layer Panel > Select Layer* and *Layer Panel > Duplicate Layer* share similar actions in the former part of the operation in Figure 3.7. Simply using Difference Filtering leads to a confusion on $\overrightarrow{Mask}$ in the first and third row. Based on this observation, we propose to compute bi-directional context that preserves temporal context information both before and after the current frame. As shown in Figure 3.7, $\overrightarrow{Mask_t}$ and $\overrightarrow{Mask_{t+1}}$ are obtained using Difference Filtering introduced above. These two masks are combined together to obtain a $\overleftarrow{Mask}$ which preserves bi-direction temporal information. We show the classification difference with and without bi-direction context in Figure 3.10, where class 0 and 12 denotes class *Layer Panel > Select Layer* and *Layer Panel > Duplicate Layer* respectively. The left side figure shows that a large proportion of class 12 video clips are miss-categorized into class 0 due to information loss caused by Difference Filtering. The right-side figure demonstrates that adding bi-direction context can effectively relieve this problem.

As described above, our Attention-aware Filtering uses Difference Filtering with Bi-direction Context. It can find the main active region as well as examine the temporal context in both forward and backward direction. We demonstrate in Section 3.6.1 that

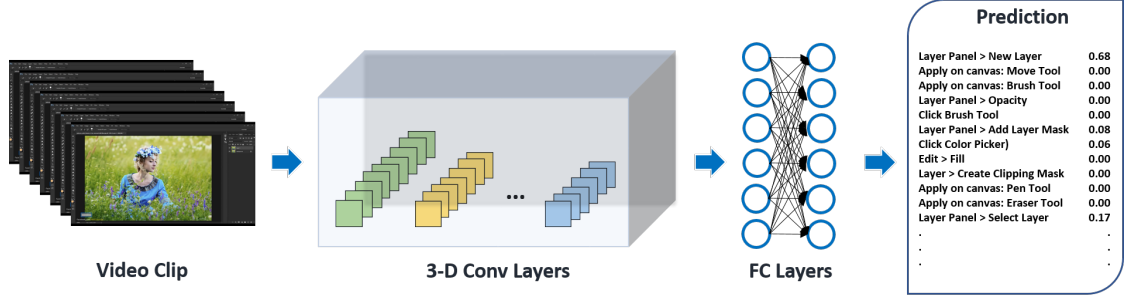


Figure 3.8: Network structure.

this process is a significant step in the proposed algorithm.

3.5.2 Video Regularization

All the video sequences are processed to the same size (in both spatial and temporal dimension) for convenience during training and testing in the proposed framework. Unlike images which can easily be resized to a fixed size, videos vary a lot in temporal extent (the first row in Figure 3.6), especially for the proposed PSOV dataset. To regularize the Photoshop command video clips, one significant point is to select frames which keep the most important information. We note that simple uniform sampling can miss such important information severely in the PSOV dataset, for that the key moments (frames that determines the command like clicking a button) distribute randomly in each video. Therefore, we take advantage of the attention area in Section 3.5.1. Redundant frames with no information left after attention filtering are removed from the video. During down-sampling, we start by taking away frames with less information; when it comes to up-sampling, we simply pad the video via random duplicating.

3.5.3 3-D CNN

Our network is modified from [90], with 5 convolution layers, two fully connected layers, and one classification output layer. Figure 3.8 shows the structure of our proposed network. Different from 2-D CNNs, 3-D network does convolution and pooling in 2-D surface and an additional time dimension, so sizes and strides in this network have three parameters: width, height and time. Our detailed parameter settings are as

Table 3.1: Ablation study. This table shows the performance for different combinations of components in the proposed framework, where 3D-Conv, Diff-Filter, Bi-Context, DataAug denote 3-D Convolution, Difference Filtering, Bi-direction Context, Data Augmentation respectively (details in Section 3.6.1).

	3-D CNN	Diff-Filter	Bi-Context	DataAug	acc-50	acc-20
2-D CNN		✓			17.02	23.18
2-D CNN*		✓	✓		19.15	24.70
RGB input	✓				51.17	54.66
3-D CNN	✓	✓			63.15	69.14
3-D CNN*	✓	✓	✓		63.76	69.73
Ours	✓	✓	✓	✓	66.37	74.97

follow: 1) convolution layers all have $3 \times 3 \times 3$ kernels with $2 \times 2 \times 2$ strides; 2) pooling layers have $2 \times 2 \times 2$ kernels with strides of $2 \times 2 \times 2$ (except the first pooling layer which has a $2 \times 2 \times 1$ stride, with no operations in temporal channel); 3) the two fully connected layers both have 2,048 output channels and are followed by drop-out layers.

The training process is done with Pytorch on a 12G TitanX GPU. The proposed network is trained from scratch, using SGD optimizer with learning rate of $1e-5$, and momentum of 0.9. Videos are regularized to 100×100 pixels in spatial domain and 50 frames in the temporal domain. It takes about 250 epochs to reach convergence with batch size of 10. Evaluation of the proposed network can be found in Section 3.6.

3.5.4 Data Augmentation

Data augmentation is widely used in various computer vision fields [22, 91, 48]. It can help introduce more diversity and make up for the data imbalance among different classes (Figure 3.4). Although the PSOV is a large dataset, data augmentation is still helpful for network training. We use the following sets of data augmentation methods to augment training data: 1) image enhancement, where we adjust the brightness, saturation, contrast, and sharpness of each video frame to augment training data, (frames within one command period have the same augmentation setting); 2) noise, where we

randomly add two kinds of noises to each frame during training: the Gaussian white noise and the salt and pepper noise; 3) translation, where we add a bit of movement to frames and use neighboring pixels to compensate for the corresponding blank area.

3.6 Experiments

We carry out extensive experiments and analysis on the PSOV dataset. First, we validate the effectiveness of each method component. Then we test the influence of command duration variance, filtered area, and analyze the confusion matrix. Through these experiments, we demonstrate that the PSOV dataset is sufficient to support deep learning, and hope to encourage better understanding and usage of this dataset.

3.6.1 Ablation Study

First, we validate the necessity and effectiveness of each component in the proposed algorithm via ablation study on the PSOV dataset. The proposed framework mainly have the following components:

- **3D-Conv**, 3-D Convolution, without which network does calculations in space domain only without temporal dimension;
- **Diff-Filter**, Difference Filtering, primary step in Attention-aware Filtering, without which network takes in original RGB images;
- **Bi-Context**, Bi-direction Context, without which network only uses Difference Filtering (in Section 3.5);
- **DataAug**, Data Augmentation, without which network does not use data augmentation during training.

We implement methods with different sets of components, and compare their performance for classifying top-20 and top-50 commands respectively. Note for top-20 command classification, networks are trained with top-20 classes on the training set; while for top-50 classification, networks are trained with top-50 classes.

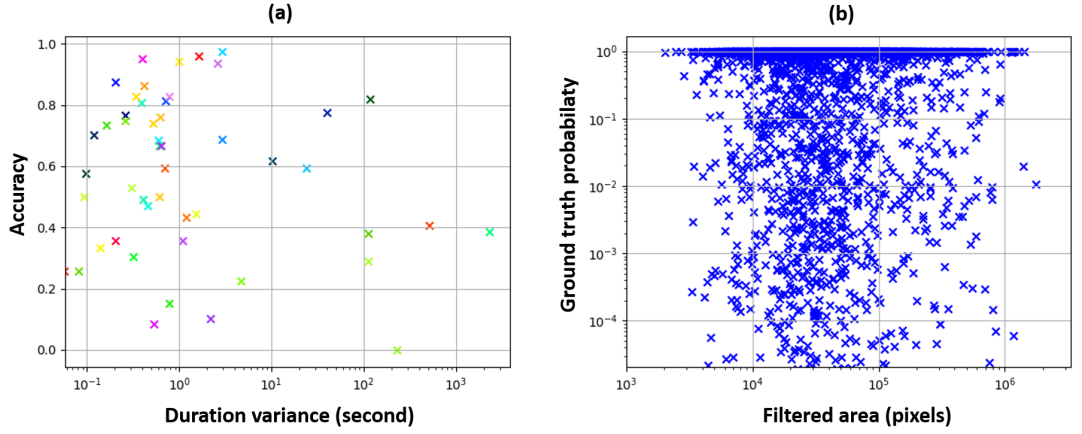


Figure 3.9: Accuracy distribution. Image (a) shows the per-class accuracy distribution on training set class duration variance; image (b) shows the per-sequence prediction versus averaged attention filter area curve. See details in Section 3.6.2.

Table 3.1 shows the statistical results, where components used in each method are denoted by check-marks. We observe that 3-D Convolution contributes largely (3-D CNN vs 2-D CNN), improving the accuracy by more than 40%. It demonstrates that temporal information is essential in recognizing Photoshop operations. The Difference Filtering (Diff-Filter) and Bi-direction Context (Bi-Context) also consistently improve performance by about 10% and 1%, illustrating the effectiveness of our Attention-aware Filtering step (Diff-Filter+Bi-Context). We also evaluate the data augmentation (DataAug) step, and find a 2%-5% improvement (3D CNN* vs Ours) in top-50 and top-20 respectively, proving that data augmentation helps on the PSOV dataset.

3.6.2 Analysis on the Command Classification Task

We draw the figure of per-class accuracy versus class duration variance in training set (image (a) in Figure 3.9). This figure shows that command sequences with extremely large duration variance (right side of the image) tend to be hard to classify (the five points in the bottom-right corner; while other videos with smaller duration variance ($< 1s$) do not have clear correspondence between accuracy and variance. It demonstrates that the

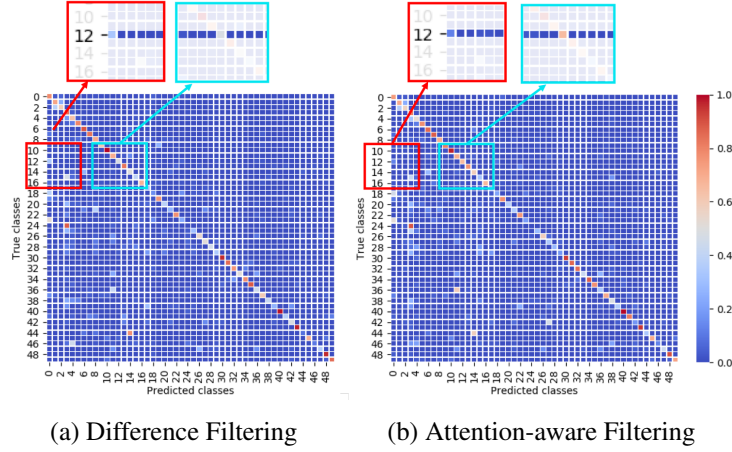


Figure 3.10: Confusion matrix for Difference Filtering and Attention-aware Filtering. Index of each row and column denotes a top-50 class; color blue to red in each pixel indicates the proportion of its row-class classified into column-class; class 0 and 12 are *Layer Panel > Select Layer* and *Layer Panel > Duplicate Layer* respectively.

proposed network can handle time variance in the PSOV dataset to a large extent.

We also draw the true class prediction probability of each sequence with the averaged pixel number of our Attention-aware Filtering to see whether the filtered motion area is related to classification difficulty. Image (b) in Figure 3.9 shows the results of this distribution, illustrating that the proposed algorithm is robust to motion area.

Figure 3.10 shows two confusion matrix for with and without Bi-direction Context, respectively. Index of each row or column represents a top-50 class. Color blue to red in each pixel indicates the proportion of its row-class classified into column-class. Figure 3.10 shows an intuitive inter-class similarity (similar pairs like class 0 and 12, class 14 and 44), indicating that bi-direction context helps in correcting wrong predictions. For example, class 0 (*Layer Panel > Select Layer*) and class 12 (*Layer Panel > Duplicate Layer*) are largely misclassified with using only Difference Filtering, but the miss-classifications are corrected via adding bi-direction context (the Attention-aware Filtering). Detailed explanation of how this happens is in Section 3.5.1 and Figure 3.7.

3.7 Summary

In this work, we present the PSOV dataset, a novel, large-scale, densely-annotated, Photoshop Operation Video dataset. The PSOV dataset consists of 564 videos with 29,204 dense annotations. To the best of our knowledge, it is the first real-world software operation dataset with large amount of videos and detailed labeling. We believe that this database can fuel researches in software intelligence, e.g. instruction video mining, autonomous software component, etc. To have a better insight into the PSOV dataset, we also propose a baseline algorithm for the command classification task. By experimenting with the proposed framework, we 1) validate that the PSOV dataset has sufficient data quantity for deep learning, 2) evaluate the effectiveness of each algorithm component, and 3) encourage better understanding and usage of the database.

Chapter 4

Progressive Domain Adaptation for Object Detection

4.1 Introduction

Object detection is an important computer vision task that aims to localize and classify objects in the images. Recent advancement in deep neural networks has brought significant improvement to the performance of object detection. However, such deep models usually require a large-scale annotated dataset for supervised learning and do not generalize well when the training and testing domains are different. For instance, the domains can differ in sceneries, weather, lighting conditions and the image appearance with respect to the camera being used. Such domain discrepancy or domain-shift causes unfavorable model generalization issues. Although additional training data from the new distribution can be added to improve the performance, collecting annotations is usually time-consuming and labor-intensive.

Unsupervised domain adaptation methods are developed to solve the domain-shift problem without using ground truth labels in the target domain. Given only available source domain annotations, the goal is to align the source and target distributions in an unsupervised manner, so that the model can also generalize to the target data. Numerous methods are developed in the context of image classification [87, 57, 58, 79, 36, 86, 28, 6], while fewer efforts have been made on more complicated tasks such as semantic

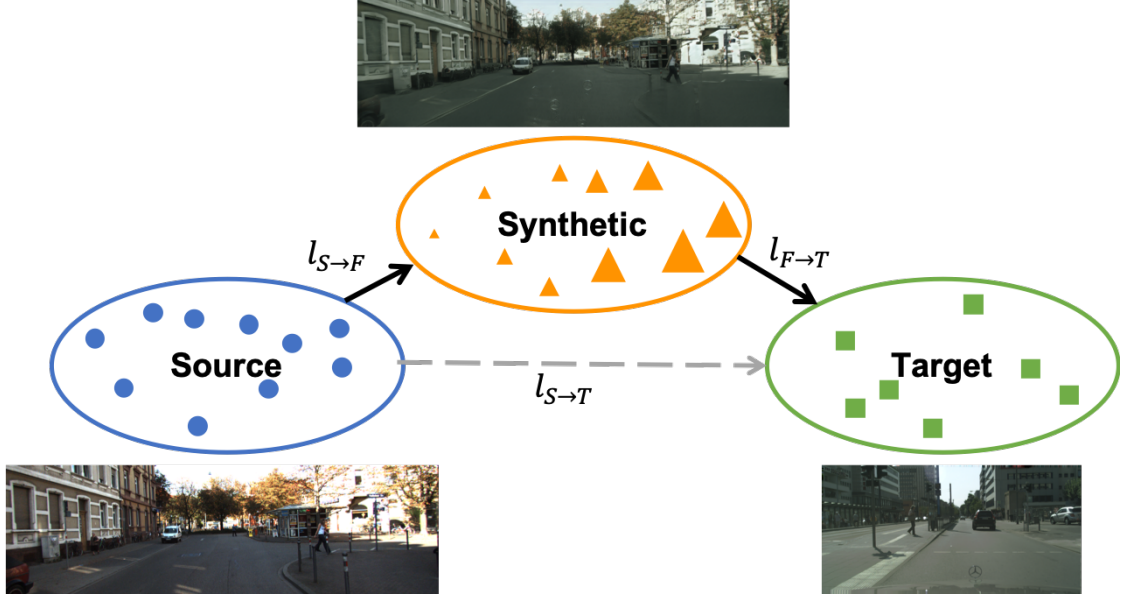


Figure 4.1: An illustration of our progressive adaptation method. Conventional domain adaptation aims to solve domain-shift problem from source to target domain, which is denoted as $l_{S \rightarrow T}$. We propose to bridge this gap with an intermediate synthetic domain that allows us to gradually solve separate subtasks with smaller gaps (shown as $l_{S \rightarrow F}$ and $l_{F \rightarrow T}$). In addition, we treat each image in the synthetic domain unequally based on its quality with respect to the target domain, where the larger size in yellow triangle stands for larger weights (i.e., the closer to the target, the higher of the weight).

segmentation [40, 85] and object detection [38, 14, 43]. In fact, such domain adaptation tasks are quite challenging as there usually exists a significant gap between source and target datasets.

In this work, we aim to minimize the efforts to align this gap across domains. Inspired by [36] that resolves the domain-shift problem via intermediate representations, we utilize an intermediate domain that lies between the source and target ones, and hence avoid direct mapping across the two distributions with a significant gap (as illustrated in Figure 4.1). Specifically, we first apply an image-to-image translation network [99] to obtain this intermediate domain via translating source images to be alike the target ones (i.e., synthetic target images). We then construct an intermediate feature space by aligning the source distributions with the intermediate ones which is an easier task than

aligning to the final target one. Once this intermediate domain is aligned, we use it as a bridge to further connect to the target domain. As a result, via the proposed progressive adaptation through the intermediate domain, the original alignment between source and target domains is decomposed into two subtasks that both solve an easier problem with a smaller domain gap.

During the alignment process, since the intermediate space is constructed in an unsupervised manner, one potential issue is that each synthetic target image may contribute unequally based on the quality of the translation. To reduce the impact of these low-quality translated images as the outliers, we propose a weighted version in our adaptation method, in which the weight can be defined based on the distance to the target distribution. That is, the image with a smaller distance to the target domain should be considered as a more important sample. In practice, we obtain this distance from the discriminator in the image translation model and incorporate it into the detection framework as a weight in the task loss.

We evaluate our method on various adaptation scenarios using numerous datasets, including KITTI [30], Cityscapes [18], Foggy Cityscapes [69] and BDD100k [95]. Experiments are conducted on domain discrepancy issues such as weather changes, camera differences and adaptation to a large-scale dataset. With the proposed progressive adaptation via the intermediate domain and the weighted importance of synthetic target samples, we show that our method performs favorably against the state-of-the-art algorithm in terms of accuracy on the target domain. The main contributions of the work are summarized as follows: 1) we introduce an intermediate domain in the proposed domain adaptation framework to achieve progressive feature distribution alignment for object detection, 2) we develop a weighted task loss during domain alignment based on the importance of the samples in the intermediate domain, and 3) we conduct extensive adaptation experiments under various object detection scenarios and achieve state-of-the-art performance.

4.2 Related Work

Object Detection. Recently, state-of-the-art object detection methods are predominately based on the deep convolutional neural networks (CNNs). These methods can be categorized into **region proposal** based and **single-shot** based by the network forwarding pipeline.

Region proposal based methods [34, 65] perform predictions on selected candidate regions. Fast R-CNN [34] extracts CNN features of a given image and computes candidate object regions via selective search [88]. The proposals are then used to crop the corresponding regions in the CNN feature map and forward to the classification and bounding box regression branches for prediction output. Extensively, Faster R-CNN [65] uses the same feature sharing idea and proposes to learn a Region Proposal Network (RPN) which utilizes the same CNN feature map to learn the candidate proposals. The PRN replaces the selective search method in Fast R-CNN, accelerating the proposal generation process.

To further reduce the computational need of proposal generation, single-shot approaches [62, 63, 64, 54] employ a fixed set of predefined anchor boxes as proposals and directly predict the category and the offset for each anchor box. The family of YOLO networks [62, 63, 64] uses a backbone network, Darknet, to extract image features. In the first version of YOLO network, the anchor box dimensions are hand picked and may not be suitable for the network to predict correct bounding boxes. YOLO v2 [63] runs k-means clustering over the bounding boxes in the training set to find better priors, using the top k box dimensions as the anchor box dimensions. An improved version of Darknet includes skip connections and is used in [64] to enhance the detection performance. Single shot detector [54] proposes to output predictions at multiple feature map levels to handle objects with different scales.

Although these methods achieve state-of-the-art performance, such success hinges on the substantial amount of labeled training data which requires high labor cost. Also, these methods would often overfit on the training domain, making them difficult to generalize to many real-world scenarios. As a result, the vision community has recently started showing great interest in employing domain adaptation techniques to object detection.

Domain Adaptation. Domain adaptation techniques aim to tackle the domain-shift between the source and target domains with unlabeled or weakly labeled images in the target domain. In recent years, adversarial learning has played a vital role in domain adaptation methods. Since the emergence of the Domain Adversarial Neural Network (DANN) [28], numerous works [6, 86, 14] have been proposed to utilize adversarial learning on the intermediate feature representations in order to align the feature distribution between two domains. Instead of performing alignment in the feature space, Tsai *et al.* [85] adopt adversarial learning in the structured output space for solving domain adaptation on semantic segmentation. Furthermore, several methods attempt to solve the problem via aligning in the pixel space, based on the unpaired image-to-image translation [99] approaches. To tackle the domain-shift of image classification, PixelDA [5] synthesizes additional images in the target domain by learning one-to-many mapping. For semantic segmentation, CyCADA [39] and AugGAN [41] both design a CycleGAN [99]-like network to transform the images from the source domain to the target domain. The transformed images are then treated as simulated training images for the target domain with the same label mapped from the source domain.

To address the domain adaptation for object detection, LSDA [38] finetunes a fully-supervised classification model for object detection with limited bounding box resources. Alternatively, Naoto *et al.* [43] train the network with synthetic data and finetune with target domain pseudo-labels. Chen *et al.* [14] attempt to approach the domain-shift issue with the unsupervised domain adaptation setting. They propose to close the domain gap on both image level and instance level via adversarial learning. In this work, we observe that simply applying adversarial learning may not be sufficient since the distance between the source and target feature distributions is often too large for one-step alignment. To address this issue, we introduce an intermediate domain which reduces the effort for mapping two significantly different distributions.

4.3 Progressive Domain Adaptation

We propose to decompose the domain adaptation problem into smaller subtasks, bridged by a synthetic domain sitting in between the source and target distribution. Tak-

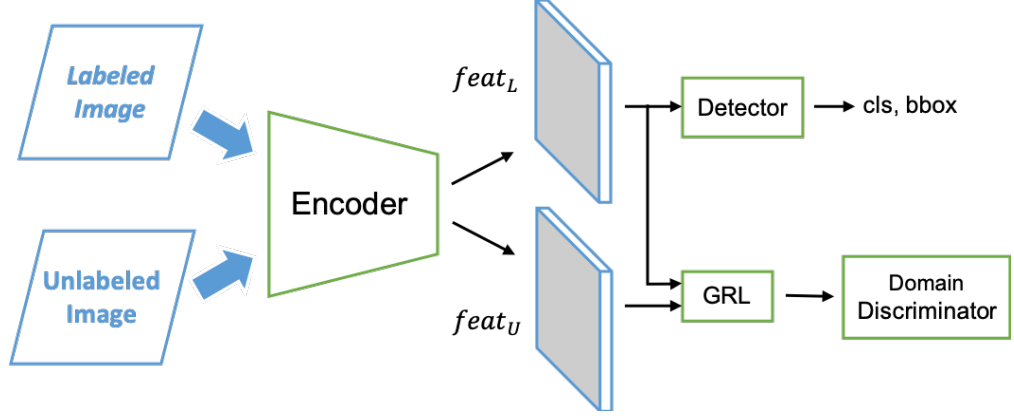


Figure 4.2: The overall structure of our adaptation network. Labeled and unlabeled images are both passed through the encoder network E to extract CNN features $feat_L$ and $feat_U$. We then use them to: 1) learn supervised object detection with the detector network from $feat_L$, and 2) forward both features to GRL and a domain discriminator, learning domain-invariant features in an adversarial manner. Our progressive adaptation framework is shown in Figure 4.3.

ing advantage of this synthetic domain, we propose to adopt a progressive adaptation strategy by closing the gap gradually through this intermediate domain. In this work, we denote the source, synthetic, and target domains as S , F and T , respectively. The conventional adaptation from a labeled domain S to the unlabeled domain T is denoted as $S \rightarrow T$, while our adaptation subtasks are expressed as $S \rightarrow F$ and $F \rightarrow T$. An overview of our progressive adaptation framework is shown in Figure 4.3. We will discuss the details of the proposed adaptation network and progressive learning in the following sections.

4.3.1 Adaptation in the Feature Space

In order to align distributions in the feature space, we build a deep model which consists of two components; one is the detection network and the other is a discriminator network for feature alignment via adversarial learning.

Detection Network. We adopt the Faster R-CNN [65] framework for the object detection task, where the detector has a base encoder network E to extract image features. Given an image I , the feature map $E(I)$ is extracted and then fed into two branches: Region Proposal Network (RPN) and a Region of Interest (ROI) classifier. We refer to these branches as the detector, which is consistent with the illustration in Figure 4.2. To train the detection network, the loss function $\mathcal{L}_{det}$ is defined as:

$$\mathcal{L}_{det}(I) = \mathcal{L}_{rpn} + \mathcal{L}_{cls} + \mathcal{L}_{reg}, \quad (4.1)$$

where $\mathcal{L}_{rpn}$, $\mathcal{L}_{cls}$, and $\mathcal{L}_{reg}$ are the loss functions for the RPN, classifier and bounding box regression, respectively. The RPN loss function also includes classification and bounding box regression similar to the ROI classifier in order to learn the candidate proposals. Different from the ROI classifier that learns a multi-class loss, the RPN network defines a binary classification loss to identify the probability of each proposal being an object; 1 being positive and 0 being negative. While $\mathcal{L}_{cls}$ is a cross-entropy loss over multiple output categories, the RPN classification loss is a binary cross-entropy loss over object and non-object. In addition, the bounding box regression loss in the RPN is only computed when the ground truth of the proposal is positive. Both of the bounding box regression loss in the RPN and ROI classifier is a *smooth* L_1 loss proposed in the Fast R-CNN [34]. Further details for training the Faster R-CNN detection network can be found in the original paper [65].

Domain Discriminator. To align the distributions across two domains, we append a domain discriminator D after the encoder E . The main objective of this branch is to discriminate whether the feature $E(I)$ is from the source or the target domain. Through this discriminator, a domain classification output is produced as $P = D(E(I)) \in \mathbb{R}^{H \times W}$. We then apply a binary cross-entropy loss to P based on the domain label d of the input image, where images from the source distributions are given the label $d = 0$ and the target images receive label $d = 1$. The discriminator loss function $\mathcal{L}_{disc}$ can be formulated as:

$$\mathcal{L}_{disc}(E(I)) = - \sum_{h,w} d \log P^{(h,w)} + (1 - d) \log(1 - P^{(h,w)}). \quad (4.2)$$

Adversarial Learning. Adversarial learning is achieved using the Gradient Reverse Layer (GRL) proposed in [27] to learn the domain-invariant feature $E(I)$. GRL is placed in between the discriminator and the detection network, only affecting the gradient computation in the backward pass. During backpropagation, the GRL will negate the gradients that flow through. As a result, the encoder E will receive gradients that force it to update in an opposite direction which maximizes the discriminator loss. This allows E to produce similar features that fools the discriminator while D tries to distinguish the image domain. For the domain adaptation task $S \rightarrow T$, given the source images I_S and target images I_T , the overall min-max loss function of the adaptive detection model is defined as the following:

$$\min_E \max_D \mathcal{L}(I_S, I_T) = \mathcal{L}_{det}(I_S) + \lambda_{disc} [\mathcal{L}_{disc}(E(I_S)) + \mathcal{L}_{disc}(E(I_T))], \quad (4.3)$$

where λ_{disc} is a weight applied to the discriminator loss that balances the detection loss.

4.3.2 Progressive Adaptation

Aligning feature distributions between two distant domains is challenging, and hence we introduce an intermediate feature space to make the adaptation task easier. That is, without directly solving the gap between the source and the target domains, we progressively perform adaptation to the target domain bridged by the intermediate domain.

Intermediate Domain. The intermediate domain is constructed from the source domain images to synthesize the target distributions on the pixel-level. We apply an image-to-image translation network, CycleGAN [99], to learn a function that maps the source domain images to the target ones, and vice versa. Since ground truth labels are only available in the source domain, we only consider the translation from source images to the target domain (i.e., synthetic target images) after the training of CycleGAN.

Synthetic target images have been utilized to assist with domain adaptation tasks [5, 41, 43] as additionally augmented target training data. Different from these approaches, we define this set of synthetic images as an individual domain F to connect both the source and target domains. One motivation behind this is that the similarity between source domain S and F is the image content, only diverging in the visual appearances, while F and the target domain T are different in image details but have similar

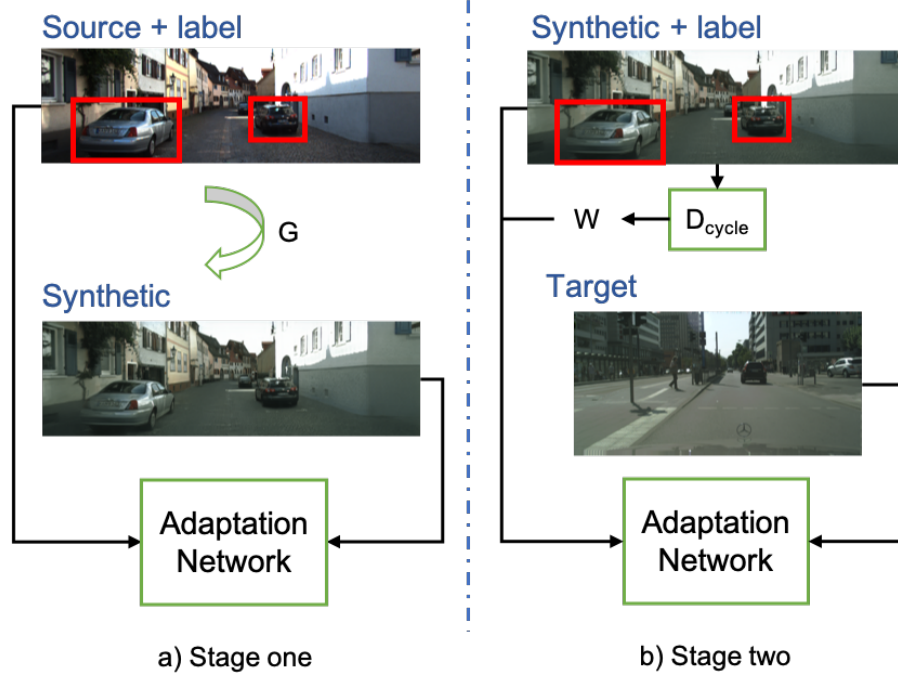


Figure 4.3: The proposed progressive adaptation framework. The algorithm includes two stages of adaptation as shown in a) and b). In a), we first transform source images to generate synthetic ones by using the generator G learned via CycleGAN [99]. Afterward, we use the labeled source domain and perform first stage adaptation to the synthetic domain. Then in b), our model applies a second stage adaptation which takes the synthetic domain with labels inherited from the source and aligns the synthetic domain features with the target distribution. In addition, a weight w is obtained from the discriminator D_{cycle} in CycleGAN to balance the synthetic image qualities in the detection loss. The adaptation network is shown in Figure 4.2.

distributions on the pixel-level. Consequently, this synthetic domain “sits” in between the source and target domains and thus can help reduce the adaptation difficulty of a large domain gap. Figure 4.4 is a feature space visualization example using the KITTI and Cityscapes datasets. This figure shows a distribution plot by mapping the features from $E(I)$ to a low dimensional 2-D space via t-SNE [89]. The plot demonstrates that in the feature space, the synthetic domain (blue) is located in between the KITTI (red) and Cityscapes (green) distributions.

Adaptation Process. Domain adaptation involves obtaining knowledge from a labeled source domain S then apply that to an unlabeled target domain T by aligning the two distributions, solving the adaptation task $S \rightarrow T$. To take advantage of the intermediate feature space during alignment, our algorithm takes incremental steps and decomposes the problem into two stages: $S \rightarrow F$ and $F \rightarrow T$, as shown in Figure 4.3 a) and b). At the first stage, we use S as the labeled domain adapting to F without labels. Due to the underlying similarity between S and F in image contents, the network focuses on aligning the feature distributions regarding the different appearances on the pixel-level. After finishing aligning the pixel discrepancies, our network takes F as the source domain for supervision and adapts to T as stage two of our method. During this step, the model is able to take the achievement via pixel-level alignment from the previous step and focus on adapting the image contents. In summary, our progressive learning separates the adaptation task into two subtasks and pays more attention to individual discrepancies during each adaptation stage.

Weighted Supervision. During the generation of synthetic target images, we observe that the quality differs in a wide range. For instance, some images fail to preserve details of objects or contain artifacts when translated, in which these failure cases may actually have a larger distance to the target distribution (see Figure 4.5 for an example). This phenomenon can be also visualized in the feature space in Figure 4.4, where some blue dots are far away from both the source and target domains.

As a result, when performing supervised detection learning on F during $F \rightarrow T$, these defects may cause confusions to our detection model, leading to false feature alignment across domains. To alleviate this problem, we propose an importance weighting strategy for synthetic samples based on their distances to the target distribution. Specifically, synthetic outliers that are farther away from the target distributions will receive less attention than the ones that are closer to the target domain. We obtain the weights by taking the predicted output scores from the target domain discriminator D_{cycle} in CycleGAN [99]. This discriminator is trained to differentiate between the source and target images with respect to the target distribution, in which the optimal discriminator

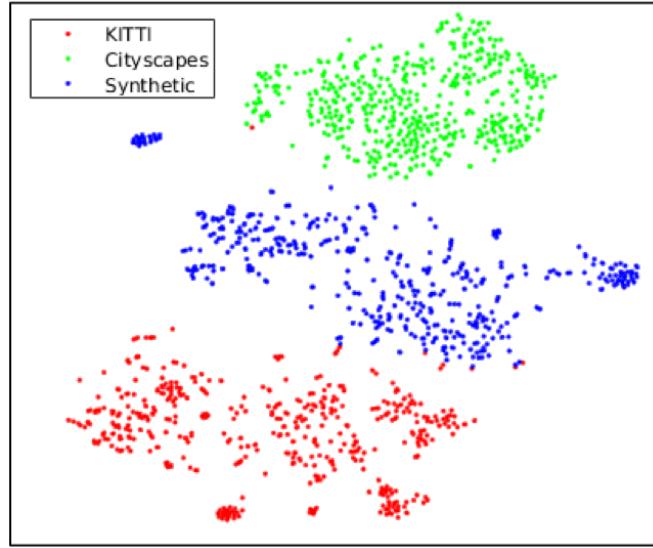


Figure 4.4: Visualization of the feature distributions via t-SNE [89], showing that our synthetic images serve as an intermediate feature space between the source and target distributions. Each dot represents one image feature extracted from E . We take 500 images from the Cityscapes validation set and 500 from the KITTI training set for comparison.



Figure 4.5: Image quality examples from the KITTI dataset synthesized to be in the Cityscapes domain. a) shows the ones that are translated with better quality. Images in b) contain artifacts and fail to preserve details of the car, almost blending into the background.

is obtained with:

$$D_{cycle}^*(I) = \frac{p_t(I)}{p_s(I) + p_t(I)}, \quad (4.4)$$

where I is the synthetic target image generated via CycleGAN, and $p_t(I)$ and $p_s(I)$ are the probability of I belonging to the source and the target domain, respectively. Here, the higher score of $D_{cycle}(I)$ represents a closer distribution to the target domain and thus should provide a higher weight. On the other hand, lower quality images which are further away from the target domain will be treated as outliers and receive a lower weight. For each image I , the importance weight is defined as:

$$w(I) = \begin{cases} D_{cycle}(I), & \text{if } I \in F \\ 1, & \text{otherwise.} \end{cases} \quad (4.5)$$

We then apply this weight to our detection loss functions when learning from the synthetic image annotations during the second stage. Thus, the final weighted objective function given images I_S and I_T is re-written from (4.2) as:

$$\min_E \max_D \mathcal{L}(I_S, I_T) = w(I_S) \mathcal{L}_{det}(I_S) + \lambda_{disc} [\mathcal{L}_{disc}(E(I_S)) + \mathcal{L}_{disc}(E(I_T))]. \quad (4.6)$$

4.4 Algorithm Analysis

4.4.1 Stage-one Analysis

Our stage-one adaptation aligns the source domain with the synthetic domain, solving the appearance differences between them. To show the impact on the target domain by aligning the source and synthetic domains in the first stage, we analyze the target domain performance as well as the synthetic one. In Table 4.1, we show the mean average precision (mAP) on synthetic target images and the final target domain, before and after applying our first-stage adaptation. The results indicate that, while adapting the appearance style between the source and synthetic domains, the model not only improves the accuracy on the synthetic domain but also enhances performance on the final target domain due to their pixel-level similarity.

Table 4.1: Performance of the first-stage adaptation. For each scenario, we show the mean average precision (mAP) of the synthetic and target domains, before and after applying our first-stage adaptation.

	KITTI $\rightarrow$ City		City $\rightarrow$ Foggy		City $\rightarrow$ BDD	
	synth	target	synth	target	synth	target
before	62.0	28.8	28.9	19.6	41.9	20.8
after	75.4	34.5	28.5	24.3	42.9	21.2

4.4.2 Stage-two Analysis

We provide further analysis for the second stage of our proposed progressive adaptation method, including the weight distribution and adding source data during the second stage.

Weight Distribution. During our second stage adaptation, we apply a dynamic weight to the task loss based on the CycleGAN [99] discriminator score of each synthetic image. We show the histogram plot of the weight distribution in Figure 4.6, which are the weights used for KITTI $\rightarrow$ Cityscapes. Images with lower scores are with worse synthetic quality and thus should contribute less to learning the adaptation model.

Additional Data. In our proposed method, we adapt from synthetic images to the target domain at the second stage. At this stage, we have performed alignment of the source and synthetic domains, hence we analyze the effect of including source images with synthetic ones during the second stage adaptation. In this analysis, we experiment on KITTI $\rightarrow$ Cityscapes with adding the source KITTI images during the second stage and show the results in Table 4.3. Since source images are more difficult to adapt to the target than the synthetic ones, we observe that the second-stage model after including source images can not take advantage of our first-stage alignment and thus produce worse results than using the synthetic domain alone.

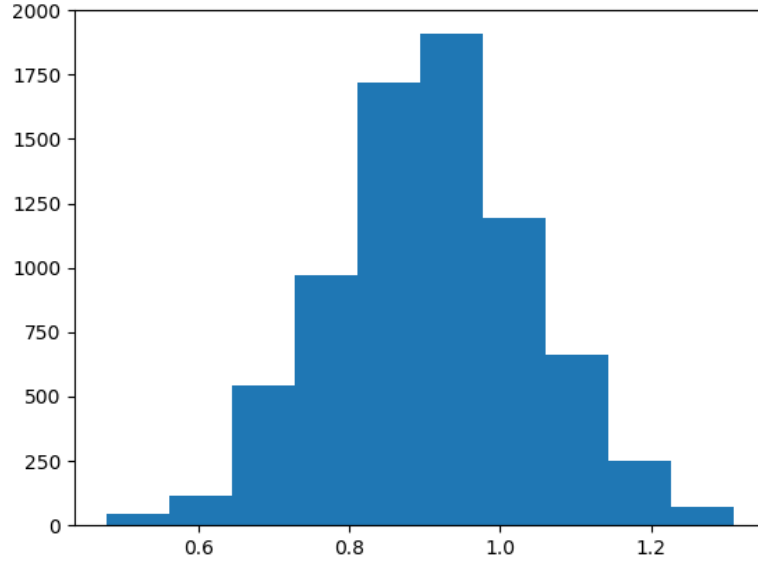


Figure 4.6: Histogram plot of the weight distribution applied to the task loss during the second stage of our method for KITTI $\rightarrow$ Cityscapes. The x-axis is the weight and the y-axis is the number of images.

Table 4.2: Performance comparison during the second stage on KITTI $\rightarrow$ Cityscapes using different data.

KITTI $\rightarrow$ Cityscapes	
Stage-two	mAP
KITTI + synth $\rightarrow$ Cityscapes	38.2
Synth $\rightarrow$ Cityscapes (Ours)	42.2

4.5 Experimental Results

In this section, we validate our method by evaluating the performance in three real-world scenarios: 1) cross camera adaptation which can cause major visual discrepancies between domains, 2) weather adaptation, and 3) adaptation from smaller annotated dataset to a much larger and diverse dataset. Figure 4.10, 4.11 and 4.12 shows exam-

ples of the detection results from the three tasks before and after applying our domain adaptation method.

For each adaptation scenario, we show a baseline Faster R-CNN result trained on the source data without applying domain adaptation, and a supervised model trained fully on the target domain data (oracle) to illustrate the existing gap between domains. Then we train the proposed model on the selected source and target domain to demonstrate the effectiveness of the proposed method. We also conduct ablation study to analyze the effectiveness of the individual proposed components. In the following, we will first describe the datasets we chose and discuss the experimental results of our adaptation tasks.

4.5.1 Implementation Details

Adaptation Network. In our experiments, we adopt VGG16 [74] as the backbone for the Faster R-CNN [65] detection network, following the setting in [14]. We design the discriminator network D using 4 convolutional layers with a filter of size 3×3 . The first 3 convolution layers have 64 channels, each followed by a leaky ReLU [60] with α set to 0.2. The final layer for domain classification has channel size 1 that outputs the binary domain label prediction. Our synthetic domain is generated by training CycleGAN [99] on the source and target domain images.

Training Details. Before applying the proposed adaptation method, we finetune the detection network using the source domain images with the ImageNet [20] pre-trained network. While training the adaptation model, we use all available annotations in the source domain including the training and validation set. We optimize the network using Stochastic Gradient Descent (SGD) with a learning rate of 0.001, weight decay of 0.0005 and momentum of 0.9. We implement the proposed method with Pytorch and train the networks with one GTX 1080 Ti GPU with 12 GB memory.

4.5.2 Datasets

KITTI. The KITTI dataset [30] contains images taken while driving in cities, highways, and rural areas. There are a total of 7,481 images in the training set. The dataset

is only used as the source domain in the proposed experiments, and we utilize the full training set.

Cityscapes. The Cityscapes dataset [18] is a collection of images with city street scenarios. It includes instance segmentation annotation which we transform into bounding boxes for our experiments. It contains 2,975 training images and 500 validation images. We use Cityscapes with the KITTI dataset in Section 4.5.3 to evaluate the cross camera adaptation and compare our results with the state-of-the-art method.

Foggy Cityscapes. As self-explanatory by the name, the Foggy Cityscapes dataset [69] is built upon the images in the Cityscapes dataset [18]. This dataset simulates the foggy weather using depth maps provided in Cityscapes with three levels of foggy weather. The simulation process can be found in the original paper [69]. Section 4.5.4 shows the experiments conducted on this simulated dataset for cross weather adaptation.

BDD100k. The BDD100k dataset [95] consists of 100k images which are split into training, validation, and testing sets. There are 70k training images and 10k validation images with available annotations. This dataset includes different interesting attributes; there are 6 types of weather, 6 different scenes, 3 categories for the time of day and 10 object categories with bounding box annotation. In our experiment, we extract a subset of the BDD100k with images labeled as *daytime*. It includes 36,728 training and 5,258 validation images. We use this subset to demonstrate the adaptation from a smaller dataset, Cityscapes, to a large-scale dataset using the proposed method in Section 4.5.5.

4.5.3 Cross Camera Adaptation

Different datasets exhibit distinct characteristics such as scenes, objects, and viewpoint. In addition, the underlying camera settings and mechanisms can also lead to critical differences in visual appearance as well as the image quality. These discrepancies are where the domain-shift takes place. In this experiment, we will show the adaptation between images taken from different cameras and with distinctive scenery plus content differences. The KITTI and Cityscapes datasets are used as source and target respectively to conduct the cross camera adaptation experiment. Experimental results show that our method performs favorably against the state-of-the-art method [14]

that learns to adapt in the feature space, matching the baseline performance of our own implementation denoted as Ours (w/o synthetic). The adaptation results are shown in Table 4.3, evaluated on the *car* class in terms of the average precision (AP).

In order to validate our method, we also conduct ablation studies using several settings. First, we demonstrate the benefit of utilizing information from the synthetic domain. When we directly augment synthetic data in the training set and include them in the source domain to perform feature-level adaptation, there is a 2.4% performance gain. In the proposed method, by adopting our progressive training scheme with the importance weights, we show in Table 4.3 that our model further improves the AP by 3.3%.

Table 4.3: Cross camera adaptation using KITTI and Cityscapes datasets. The results show the average precision (AP) of the *car* class shared between the two domains.

KITTI $\rightarrow$ Cityscapes	
Method	AP
Faster R-CNN	28.8
FRCNN in the wild [14]	38.5
Ours (w/o synthetic)	38.2
Ours (synthetic augment)	40.6
Ours (progressive)	43.9
Oracle	55.8

Table 4.4: Analysis of our weighted task loss compared to several arbitrary weight settings. We show that by setting each image weight with respect to the distance from the target distribution improves the model performance.

KITTI $\rightarrow$ Cityscapes						
weight	0.8	0.9	1	1.1	1.2	Ours
AP	39.8	42.8	42.2	41.1	42.6	43.9

In addition, we present the advantage of our weighted task loss in balancing the uneven synthetic image qualities. In Table 4.4, we show the analysis for using different fixed weights and our importance weighting method. When setting an arbitrary weight over the task loss, we obtain unstable results either with higher or lower weight. Our method dynamically determines the weight of each image¹ based on the distance from the target distribution and is 1.7% closer to the oracle performance after learning from such importance weighted supervision. Overall, we show that our model can reduce the domain-shift problem caused by the camera along with other content differences across two distinct datasets and achieves state-of-the-art performance.

Table 4.5: Weather adaptation focusing on clear weather to foggy weather using the Cityscapes and Foggy Cityscapes datasets respectively. Performance is evaluated using the mean average precision (mAP) across 8 classes.

Cityscapes → Foggy Cityscapes						
Category	Faster R-CNN	FRCNN in the wild [14]	Ours (w/o synthetic)	Ours (synthetic augment)	Ours (progressive)	Oracle
person	23.2	25.0	30.2	36.6	36.0	37.8
rider	29.4	31.0	37.9	45.3	45.5	48.4
car	36.9	40.5	46.1	55.0	54.4	58.8
truck	7.1	22.1	14.7	24.2	24.3	25.2
bus	17.9	35.3	26.9	43.9	44.1	53.3
train	2.4	20.2	7.0	18.5	25.8	15.8
motorcycle	13.9	20.0	20.8	28.4	29.1	35.4
bicycle	25.7	27.1	31.5	37.1	35.9	39.0
mAP	19.6	27.6	26.9	36.1	36.9	39.2

¹In this case, the averaged weight obtained from the discriminator is around 0.9.

4.5.4 Weather Adaptation

Under real-world scenarios, supervised object detection models can be applied in different weather conditions where they may not have sufficient knowledge of. However, it is difficult to obtain a large number of annotations in every weather condition for the models to learn. This section studies the weather adaptation from clear weather to a foggy environment. The Cityscapes dataset is used as the source domain and the Foggy Cityscapes dataset as the target domain.

For a fair comparison with the state-of-the-art method [14], we evaluate our method on 8 classes in the Cityscapes dataset as shown in Table 4.5. This table shows that our method can further reduce the domain gap across weather conditions. In addition, we discuss the characteristics of the two datasets and why it is in favor of our method.

When synthetic images are used during adaptation, the results show that there is a 9.2% improvement in performance. We note that the target Foggy Cityscapes dataset is fundamentally the same image as the source, Cityscapes dataset. On the other hand, our synthetic data is also generated using the same foundation which turns out to reduce the difficulty for image translation and synthesizes more target-alike images than other scenarios. Thus, the synthetic data is closely distributed to the target domain and inherits informative labels for the network to learn, enhancing performance in the target domain. Given such information learned from the synthetic domain, both our method and the synthetic augmented one climbs closely to the oracle result. Although the synthetic domain lies close to the target distribution, we show in the results that our progressive training can still assist the adaptation process, improving performance and at the same time generalizing well to different categories. To sum up, this experiment not only demonstrates the adaptation to a foggy weather condition but also shows the capability of using synthetic images to facilitate the distribution alignment process.

4.5.5 Adaptation to Large-scale Dataset

Digital cameras have developed quickly over the years and collecting a large number of images is not a difficult task in the modern world. However, labeling the collected images is a major issue when it comes to building a dataset for supervised learning meth-

Table 4.6: Adaptation from a smaller Cityscapes dataset to a larger and diverse BDD100k dataset. A subset of the BDD100k dataset labeled as *daytime* is used as the target domain. We evaluate the mean average precision (mAP) of 10 classes which are available across the two domains.

Cityscapes $\rightarrow$ BDD100k daytime					
Category	Faster R-CNN	Ours (w/o synthetic)	Ours (synthetic augment)	Ours (progressive)	Oracle
bike	19.4	20.4	23.1	25.3	36.2
bus	20.4	20.2	25.3	23.7	58.2
car	49.0	49.2	51.9	51.8	62.3
motor	17.2	16.6	15.7	16.1	36.1
person	31.1	32.1	36.0	37.6	46.2
rider	26.5	27.8	31.6	32.9	43.6
light	11.5	11.9	12.7	14.0	43.5
sign	14.6	14.9	20.8	22.2	49.7
train	0	0	0	0	0
truck	18.9	19.2	20.2	19.3	57.6
mAP	20.8	21.2	23.7	24.3	43.3

ods. In this experiment, we want to examine the adaptation from a relatively smaller dataset to a large unlabeled domain containing distinct attributes. We show that we can harvest more from existing resources and adapt them to a complicated environment. To this end, we use the Cityscapes and BDD100k datasets as the source and target domains respectively to conduct this experiment. We choose a subset of the BDD100k dataset annotated as *daytime* to be our target domain and use the city scene as the adaptation factor since there only exists daytime data in the Cityscapes dataset.

Performance is evaluated on the 10 classes in the BDD100k test set as shown in Table 4.6. From the baseline and oracle results, we can observe the difficulty and the significant performance gap between the source and target domains. Without using the synthetic data, the network has a harder time in adapting to a much diverse dataset with only 0.4% improvement after directly aligning the source and target domains. When synthetic data is introduced to the source training set, the model learns to generalize better to the target domain and increases the performance by 2.5%. Finally, our method progressively adapts to the target domain by utilizing the intermediate feature space and receives an additional 0.6% gain in accuracy compared to the direct synthetic data augmentation. We show in this experiment that our progressive adaptation can squeeze more juice out of the available knowledge and generalize better to a diverse environment, which is a critical issue in real-world applications.

4.5.6 Synthetic Images and Visual Results

Examples of the source and synthetic images in each scenario are shown in Figure 4.7, 4.8 and 4.9. In addition, more detection results before and after performing the domain adaptation are shown in Figure 4.10, 4.11 and 4.12, which are cross camera, cross weather and adaptation to large-scale dataset, respectively. The visualization results show that after performing our adaptation method, the model is able to handle challenging tasks including occlusions and distant objects.



Figure 4.7: Examples of the source and synthetic images used in KITTI $\rightarrow$ Cityscapes. Top row is the KITTI images while the bottom row is the synthetic images after being transformed.



Figure 4.8: Examples of the source and synthetic images used in Cityscapes $\rightarrow$ Foggy Cityscapes. Top row is the Cityscapes images while the bottom row is the synthetic images after being transformed.



Figure 4.9: Examples of the source and synthetic images used in Cityscapes $\rightarrow$ BDD100k daytime. Top row is the Cityscapes images while the bottom row is the synthetic images after being transformed.

4.6 Summary

In this work, we propose a progressive adaptation method that bridges the domain gap using an intermediate domain, decomposing a more difficult task into two easier subtasks with a smaller gap. We obtain the intermediate domain by transforming the source images to the target distribution. Using this domain, our method progressively solves the adaptation subtasks by first adapting from source to the intermediate domain and then finally to the target domain. In addition, we include a weighted loss during

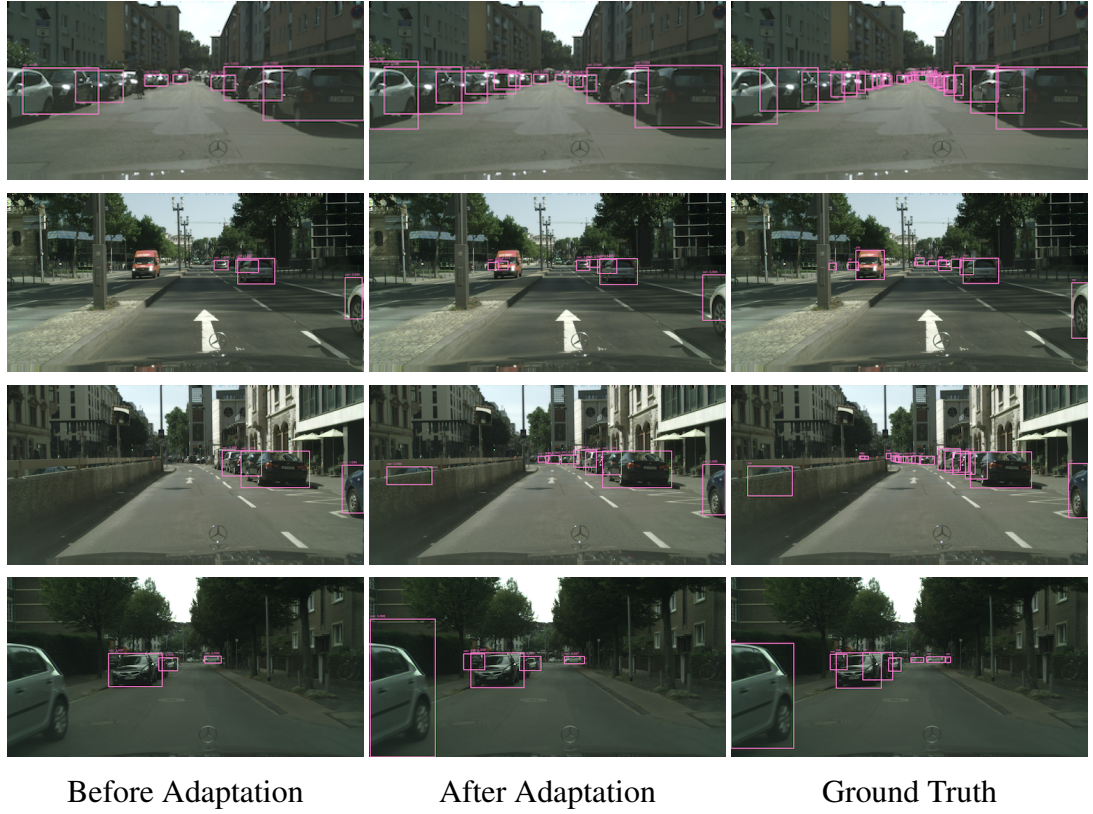


Figure 4.10: Examples of the detection results for the task KITTI $\rightarrow$ Cityscapes. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels.

stage two of our method to balance the different image qualities in the intermediate domain. Experimental results show that our method performs favorably against the state-of-the-art method and can further reduce the domain discrepancy under various scenarios, such as the cross-camera case, weather condition, and adaption to a large-scale dataset.

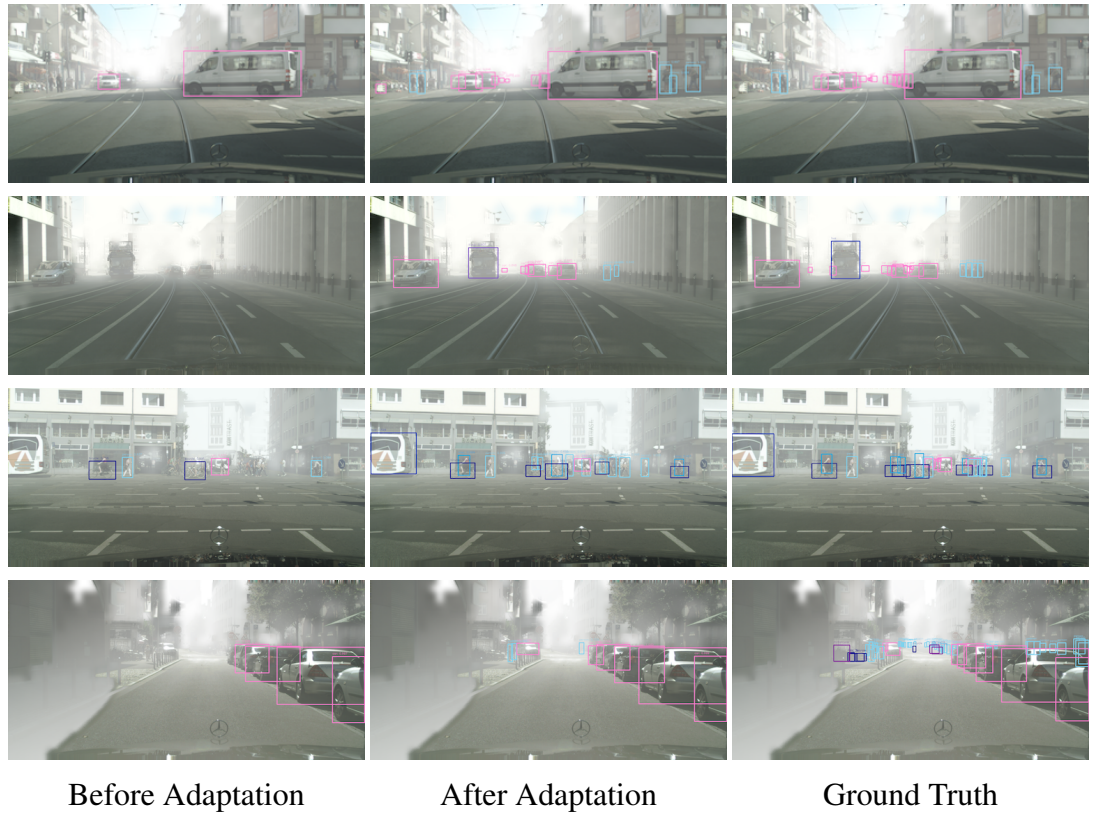


Figure 4.11: Examples of the detection results for the task Cityscapes $\rightarrow$ Foggy Cityscapes. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels.

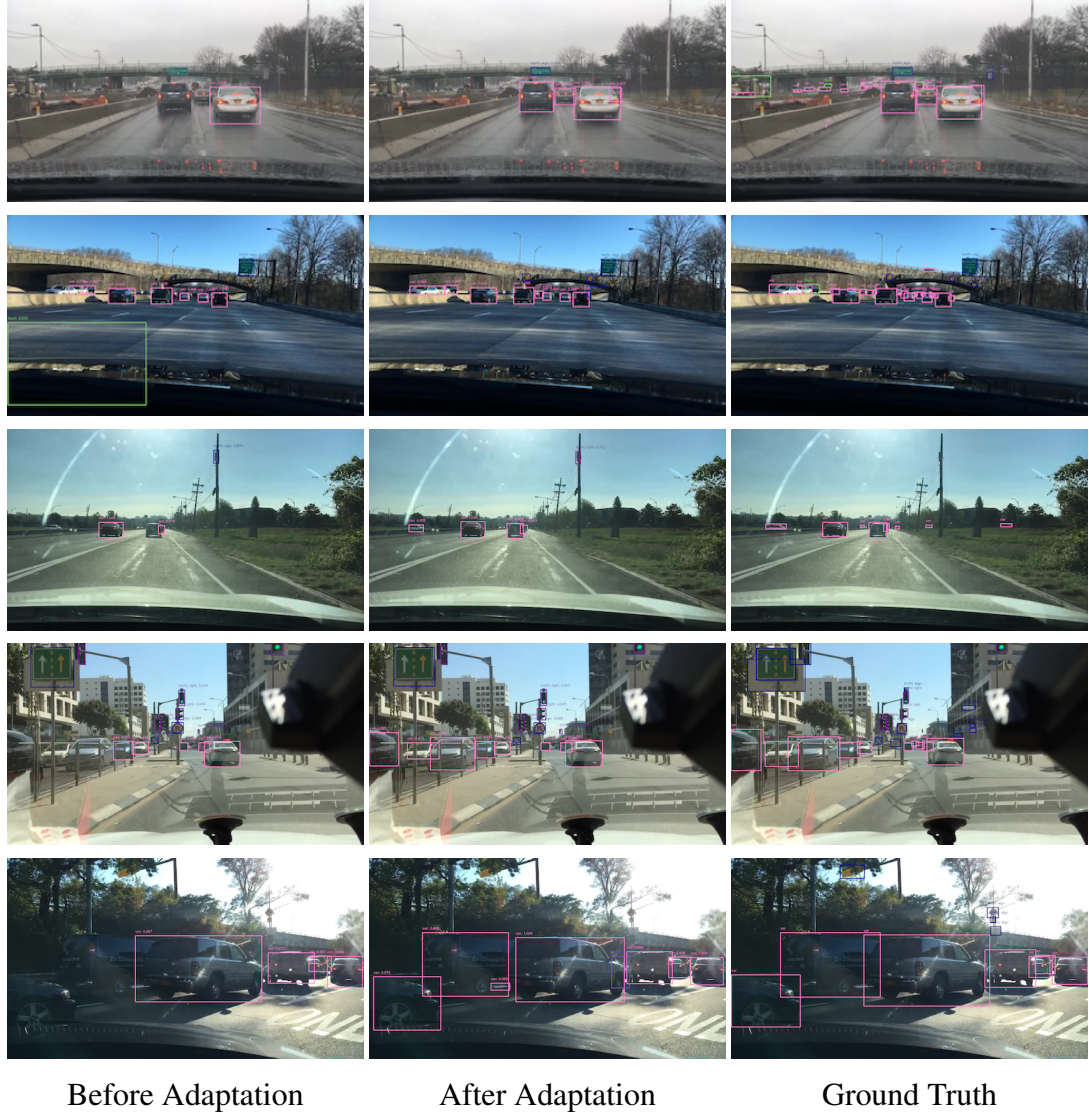


Figure 4.12: Examples of the detection results for the task Cityscapes $\rightarrow$ BDD100k daytime. We show the detection results on the target domain before and after applying our adaptation method as well as the ground truth labels.

Chapter 5

Conclusion and Future Work

In this thesis, we present algorithms extracting both spatial and temporal information from videos and also tackle the domain discrepancy issue that causes poor model generalization based on the object detection task. First of all in Chapter 2, we learn to tell the brake and turn signals. We define 8 signal classes and utilize a CNN-LSTM model to extract spatial features and learn the long-term dependencies of the image sequences. Our experimental results show the effectiveness of utilizing temporal model when recognizing the dynamic turn signals and make accurate predictions of the different signal classes.

Next in Chapter 3, we build a dataset to explore the potential of software intelligence and how digital actions differs from real-world actions. Understanding the differences of digital actions, we introduce an attention-aware preprocessing method to extract action related regions in the sequence as input for our spatial-temporal model to learn. We show in the experiments that our method assists the deep model in learning the temporal knowledge, improving the recognition accuracy. In addition, the results indicate that our PSOV dataset has sufficient quantity for deep learning methods to perform action detection or tasks like video to text.

Finally, we performed domain adaptation on the object detection task in Chapter 4. Since different domains usually have a large domain gap between them, not only is this hard for the model to generalize but also increases the difficulty in distribution alignment. To approach this problem, we propose a progressive adaptation method which bridges two domains with an intermediate domain and gradually adapt to the testing

domain. The adaptation results show that we are able to perform incremental adaptation and obtain better performance on the testing domain.

In the end, we conclude by discussing the potential directions for future research. First of all, we can expand the signal recognition framework by combining object detection into the temporal model to localize vehicles and predict rear signals in an end-to-end fashion for faster and easier inference. A second potential direction is to utilize the PSOV dataset and explore further into the digital actions towards making software intelligence possible. Potential tasks include localizing command action intervals in the tutorial video and parsing videos into step by step text instructions. Finally, the domain adaptation task can further look into the effectiveness in using multi-modal image translation to generate the synthetic domain and adapt to multiple datasets. Our progressive method can also be extended into multiple steps or utilize the synthetic source data translated from the target domain. This thesis can be a foundation for such future directions and work towards constructing a more robust system for video understanding and also improving the domain generalization of supervised models such as object detection.

Bibliography

- [1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 2010.
- [2] A. Almagambetov, M. Casares, and S. Velipasalar. Autonomous tracking of vehicle rear lights and detection of brakes and turn signals. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–7, 2012.
- [3] A. Almagambetov, S. Velipasalar, and M. Casares. Robust and computationally lightweight autonomous tracking of vehicle taillights and signal detection by embedded smart cameras. *IEEE Transactions on Industrial Electronics*, 62(6):3732–3741, 2015.
- [4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [5] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [6] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, 2016.
- [7] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool. One-shot video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [8] M. Casares, A. Almagambetov, and S. Velipasalar. A robust algorithm for the detection of vehicle turn signals and brake lights. In *IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*, pages 386–391, 2012.
- [9] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *IEEE International Conference on Computer Vision*, 2015.
- [10] D.-Y. Chen, Y.-J. Peng, L.-C. Chen, and J.-W. Hsieh. Nighttime turn signal detection by scatter modeling and reflectance-based direction recognition. *IEEE Sensors Journal*, 14(7):2317–2326, 2014.

- [11] H.-T. Chen, Y.-C. Wu, and C.-C. Hsu. Daytime preceding vehicle brake light detection using monocular vision. *IEEE Sensors Journal*, 16(1):120–131, 2016.
- [12] L.-C. Chen, J.-W. Hsieh, S.-C. Cheng, and Z.-R. Yang. Robust rear light status recognition using symmetrical surfs. In *IEEE International Conference on Intelligent Transportation Systems*, pages 2053–2058, 2015.
- [13] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [14] Y. Chen, W. Li, C. Sakaridis, D. Dai, and L. V. Gool. Domain adaptive faster r-cnn for object detection in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [15] J. Cheng, S. Liu, Y.-H. Tsai, W.-C. Hung, S. De Mello, J. Gu, J. Kautz, S. Wang, and M.-H. Yang. Learning to segment instances in videos with spatial propagation network. *arXiv preprint arXiv:1709.04609*, 2017.
- [16] J. Cheng, Y.-H. Tsai, S. Wang, and M.-H. Yang. Segflow: Joint learning for video object segmentation and optical flow. In *IEEE International Conference on Computer Vision*, 2017.
- [17] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [18] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [19] Z. Cui, S.-W. Yang, and H.-M. Tsai. A vision-based hierarchical framework for autonomous front-vehicle taillights detection and signal recognition. In *IEEE International Conference on Intelligent Transportation Systems*, pages 931–937, 2015.
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
- [21] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.
- [22] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision*, 2015.
- [23] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning*, 2017.

- [24] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 2010.
- [25] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [26] B. Frohlich, M.ENZweiler, and U. Franke. Will this car change the lane?-turn signal recognition in the frequency domain. In *IEEE Intelligent Vehicles Symposium*, pages 37–42, 2014.
- [27] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, 2015.
- [28] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 2016.
- [29] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research*, 2013.
- [30] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.
- [31] S. Gelly and D. Silver. Achieving master level play in 9 x 9 computer go. In *Association for the Advancement of Artificial Intelligence*, 2008.
- [32] S. Gelly and D. Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 2011.
- [33] R. Girdhar, D. Ramanan, A. Gupta, J. Sivic, and B. Russell. Actionvlad: Learning spatio-temporal aggregation for action classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [34] R. Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision*, 2015.
- [35] G. Gkioxari and J. Malik. Finding action tubes. *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [36] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *IEEE International Conference on Computer Vision*, 2011.
- [37] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [38] J. Hoffman, S. Guadarrama, E. S. Tzeng, R. Hu, J. Donahue, R. Girshick, T. Darrell, and K. Saenko. Lsda: Large scale detection through adaptation. In *Advances in Neural Information Processing Systems*, 2014.

- [39] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International Conference on Machine Learning*, 2018.
- [40] J. Hoffman, D. Wang, F. Yu, and T. Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *CoRR*, abs/1612.02649, 2016.
- [41] S.-W. Huang, C.-T. Lin, S.-P. Chen, Y.-Y. Wu, P.-H. Hsu, and S.-H. Lai. Auggan: Cross domain adaptation with gan-based data augmentation. In *European Conference on Computer Vision*, 2018.
- [42] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [43] N. Inoue, R. Furuta, T. Yamasaki, and K. Aizawa. Cross-domain weakly-supervised object detection through progressive domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [44] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.
- [45] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*, pages 675–678, 2014.
- [46] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
- [47] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
- [48] A. Khoreva, R. Benenson, E. Ilg, T. Brox, and B. Schiele. Lucid data dreaming for multiple object tracking. *arXiv preprint arXiv:1703.09554*, 2017.
- [49] M. Kim, S. Kim, S. Park, M.-T. Choi, M. Kim, and H. Goma. Service robot for the elderly. *IEEE Robotics & Automation Magazine*, 2009.
- [50] S. Lefèvre, A. Carvalho, Y. Gao, H. E. Tseng, and F. Borrelli. Driver models for personalised driving assistance. *Vehicle System Dynamics*, 2015.
- [51] Y. Li, Z.-x. Cai, and J. Tang. Recognition algorithm for turn light of front vehicle. *Journal of Central South University*, 19(2):522–526, 2012.
- [52] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, 2014.
- [53] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):978–994, 2011.

- [54] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision*, 2016.
- [55] W. Liu, H. Bao, J. Zhang, and C. Xu. Vision-based method for forward vehicle brake lights recognition. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 8(6):167–180, 2015.
- [56] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.
- [57] M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, 2015.
- [58] M. Long, J. Wang, and M. I. Jordan. Deep transfer learning with joint adaptation networks. In *International Conference on Machine Learning*, 2017.
- [59] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.
- [60] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.
- [61] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, 2015.
- [62] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [63] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [64] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.
- [65] 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 and Machine Intelligence*, 2017.
- [66] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [67] C. Rhee, W. Chung, M. Kim, Y. Shim, and H. Lee. Door opening control using the multi-fingered robotic hand for the indoor service robot. In *IEEE International Conference on Robotics and Automation*, 2004.

- [68] M. D. Rodriguez, J. Ahmed, and M. Shah. Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [69] C. Sakaridis, D. Dai, and L. V. Gool. Semantic foggy scene understanding with synthetic data. *International Journal of Computer Vision*, 2018.
- [70] A. Shashua, Y. Gdalyahu, and G. Hayun. Pedestrian detection for driving assistance systems: Single-frame classification and system level performance. In *IEEE Intelligent Vehicles Symposium*, 2004.
- [71] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, 2017.
- [72] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 2017.
- [73] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014.
- [74] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [75] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *Association for the Advancement of Artificial Intelligence*, 2017.
- [76] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.
- [77] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, 2012.
- [78] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.
- [79] B. Sun and K. Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision Workshops*, 2016.
- [80] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [81] W. Taggart, S. Turkle, and C. D. Kidd. An interactive robot in a nursing home: Preliminary remarks. In *Cognitive Science Society Workshop*, 2005.
- [82] P. Thammakaroorn and P. Tangamchit. Predictive brake warning at night using tail-light characteristic. In *IEEE International Symposium on Industrial Electronics*, pages 217–221, 2009.

- [83] C. Tomasi and T. K. Detection. Tracking of point features. Technical report, Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, 1991.
- [84] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *IEEE International Conference on Computer Vision*, 2015.
- [85] Y.-H. Tsai, W.-C. Hung, S. Schuster, K. Sohn, M.-H. Yang, and M. Chandraker. Learning to adapt structured output space for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [86] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [87] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014.
- [88] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.
- [89] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 2008.
- [90] G. Varol, I. Laptev, and C. Schmid. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
- [91] P. Voigtlaender and B. Leibe. Online adaptation of convolutional neural networks for video object segmentation. *arXiv preprint arXiv:1706.09364*, 2017.
- [92] J. Wang, C. Xiao, T. Zhu, C.-H. Hsueh, W.-J. Tseng, and I.-C. Wu. Only-one-victor pattern learning in computer go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
- [93] J.-G. Wang, L. Zhou, Y. Pan, S. Lee, Z. Song, B. S. Han, and V. B. Saputra. Appearance-based brake-lights recognition using deep learning and vehicle detection. In *IEEE Intelligent Vehicles Symposium*, pages 815–820, 2016.
- [94] G. N. Yannakakis. Game ai revisited. In *Conference on Computing Frontiers*, 2012.
- [95] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.
- [96] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [97] W. Zaremba and I. Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

- [98] G. Zhong, Y.-H. Tsai, Y.-T. Chen, X. Mei, D. Prokhorov, M. James, and M.-H. Yang. Learning to tell brake lights with convolutional features. In *IEEE International Conference on Intelligent Transportation Systems*, pages 1558–1563, 2016.
- [99] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision*, 2017.