UNIVERSITY OF OTTAWA

MASTERS THESIS

Forward Leading Vehicle Detection for Driver Assistant System

Author:
Wen WEN

uOttawa

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Applied Science

in the

School of Electrical Engineering and Computer Science
Faculty of Engineering, University of Ottawa

Declaration of Authorship

I, Wen WEN, declare that this thesis titled, "Forward Leading Vehicle Detection for Driver Assistant System" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: 12^{th} January, 2021

UNIVERSITY OF OTTAWA

Abstract

Faculty of Engineering
School of Electrical Engineering and Computer Science

Masters of Applied Science

Forward Leading Vehicle Detection for Driver Assistant System

by Wen WEN

Keeping a safe distance from the forward-leading vehicle is an essential feature of modern Advanced Driver Assistant Systems (ADAS), especially for transportation companies with a fleet of trucks. We propose in this thesis a Forward Collision Warning (FCW) system, which collects visual information using smartphones attached for instance to the windshield of a vehicle. The basic idea is to detect the forward-leading vehicle and estimate its distance from the vehicle. Given the limited resources of computation and memory of mobile devices, the main challenge of this work is running CNN-based object detectors at real-time without hurting the performance.

In this thesis, we analyze the bounding boxes distribution of the vehicles, then propose an efficient and customized deep neural network for forward-leading vehicle detection. We apply a detection-tracking scheme to increase the frame rate of vehicle detection and maintain good performance. Then we propose a simple leading vehicle distance estimation approach for monocular cameras. With the techniques above, we build an FCW system that has low computation and memory requirements that are suitable for mobile devices. Our FCW system has 49% less allocated memory, 7.5% higher frame rate, and 21% less battery consumption speed than popular deep object detectors. A sample video is available at https://youtu.be/-ptvfabBZWA.

Acknowledgements

Fisrt of all, I would like to thank my supervisor Professor Robert Laganiere who provide me an opportunity of developing a practical FCW system. When I was heading to a wrong direction, or when I was overconfident, his guidance always appears promptly. Without his trust and patience, this work cannot be done with solid results.

Also, I would like to thank many students in our lab, such as Farzan Erlik Nowruzi, Md Atiqur Rahman, Yong Wang, Wassim El Ahmar, and Lu Sun, from whom I learned a lot. Especially, I would like to thank a co-author of my paper, Hamed Habibi Aghdam. Our work was presented on the IV2020 conference. To me, he is not only a colleague, but also a friend, and a mentor showing me how to polish my works, and how to write my first conference paper in a proper way.

Then, I thank the teammates in our newly founded robot club. They always share their thoughts to me about new computer vision technologies and ideas. Such a team gives me motivation and strengthened my confidence about computer vision.

Finally, I present tons of thanks and love to my parents who support a huge part of my spending, who relief my stress from time to time, and who trust my choice on having my career in computer vision.

Contents

List of Figures

List of Tables

List of Abbreviations

ADAS	**A**dvanced **D**riving **A**ssistance **S**ystems
AP	**A**verage **P**recision
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
FCW	**F**orward **C**ollision **W**arning
FPS	**F**rame **P**er **S**econd
GHz	**G**iga **H**ertz
GPU	**G**raphical **P**rocessing **U**nit
ILSVRC	**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
IoU	**I**ntersection **O**ver **U**nion
NMS	**N**on **M**aximum **S**uppression
ReLU	**Re**ctified **L**inear **U**nit
ROI	**R**egion **O**f **I**nterest
RPN	**R**egion **P**roposal **N**etwork

Chapter 1

Introduction

Nowadays, *Machine Learning* and *Deep Learning* have a tangible impact on nearly all aspects of our life. They have changed the common practices in many fields, such as agriculture, manufacturing, commerce, transportation, medical services, banking, and even education. Numerous researchers are trying to integrate deep learning methods into their research fields to achieve more promising results. One such example is how deep learning methods outperform various traditional techniques and have become a game-changer in the field of computer vision, with regards to many different tasks like image classification, object detection, object tracking, image segmentation, and more.

The origin of deep learning in *Artificial Neural Networks (ANN)* dates back to decades ago, when the first neural networks came out in the 1960s [28][56]. One essential component of deep learning methods is *Convolutional Neural Network (CNN)*[3]. As the memory and computation requirements of CNN models are high, researches have explored many techniques to optimize and compress CNNs. Some studies have designed new specialized hardware units to accelerate neural computation. In recent years, the computational cost has been less a problem. Many deep learning methods have been implemented onto embedded systems and mobile devices, facilitating the deployment of their daily-life systems.

Deep learning has a tremendous impact on the transportation industry in many ways. For example, by extracting high-level information from digital surveillance cameras, it is possible to make traffic flow predictions [44] and make traffic flow maps, with which drivers can choose their routes. The *Autonomous Vehicles (AV)*, a fascinating technology of the future, largely depends on the advances of deep learning [37]. Over the last few years, *Advanced Driver Assistant Systems (ADAS)* have also been designed with deep learning methods [60] to accomplish tasks such as vehicle detection, traffic signs detection and recognition, assisting changing lane, monitoring blind spot, making our driving safer.

Recently, automobile manufacturers like Volvo, Ford, and General Motors, and new players such as Tesla, Google (Project Waymo), Faraday Future, and TuSimple

are in a race to develop autonomous vehicles. While the trend of developing AV is unstoppable, ADAS is now still an essential driving safety solution before fully autonomous vehicles get accepted and applied widely. To acquire sufficient data, modern ADAS uses various types of sensors, *e.g.* color camera, infrared sensor, LIDAR, and Radar, which makes modern ADAS effective, but economically and computationally more expensive. More rudimentary ADAS systems only use limited types and numbers of sensors and provide more basic functions such as cruise control.

The motivation of this thesis comes from a commercial trucking safety project that needs an economical upgrade to vehicles' collision warning systems. The objective of this thesis is to design and build a practical collision warning system based on forward-leading vehicle detection using deep learning models. The system is designed for devices with a monocular color camera.

In this chapter, an overview of ADAS is presented in section 1.1. Section 1.2 gives a brief introduction to CNN architectures. Section 1.3 gives an overview of the object detection tasks, the challenges we have in this field, followed by the problem statement, contribution, and overall thesis structure in Section 1.5.

1.1 Advanced Driver Assistant System

The *Advanced Driver Assistant System (ADAS)* development started with the *Modern Cruise Control* function in 1948 by Ralph R. Teetor. The system was able to control the car speed automatically while driving. In the following decades, more features were added, from *Adaptive Cruise Control (ACC)*, *Anti-lock Braking System (ABS)*, *Lane Departure Warning (LDW)*, *Forward Collision Warning (FCW)*, to name a few.

In the last decades, developing ADAS has become a trend in the automotive industry. On October 19, 2002, a father was backing his car into the driveway and hit his beloved child, two-year-old Cameron Gulbransen [1], because he cannot see the child behind the vehicle. After such a tragedy, the world realized that installing ADAS in vehicles was no longer an optional move for comfort and convenience, but necessary to keep the drivers and the surrounding peoples safe. Therefore, upgrading vehicles with ADAS has been included in written acts [2], showing its importance.

[1] https://www.kidsandcars.org/child_story/cameron-gulbransen/
[2] Cameron Gulbransen Kids Transportation Safety Act of 2007

1.2 Convolutional Neural Networks

A CNN [3] is an artificial network, which takes images, audio or video sequences as input, and process those data to generate customized outputs, which could be, for instance, predicted values in regression tasks, or classes in classification tasks. Fig. 1.1 shows a typical example of a CNN for a classification task. This CNN is composed of a *convolution layer (Conv)* followed by a *Rectified Linear Units (ReLU)* function. The next layer is a *spatial pooling* layer to reduce the dimension. These two layers are fundamental for nearly all the CNN models. Usually, to increase CNN models' discriminative power, multiple groups of Conv, ReLU, and pooling are applied together. After that, the feature map is flattened to a 1-D array, sent through one or more fully connected layers for getting the final classification result. In this example, when the input image contains a car, the network should produce the class "car" as the classification result with the highest probability.

More details about the definitions and utilities of the layers in CNN structure will be explained in Section 2.1, and several state-of-the-art CNN backbones will be presented in Section 2.2.

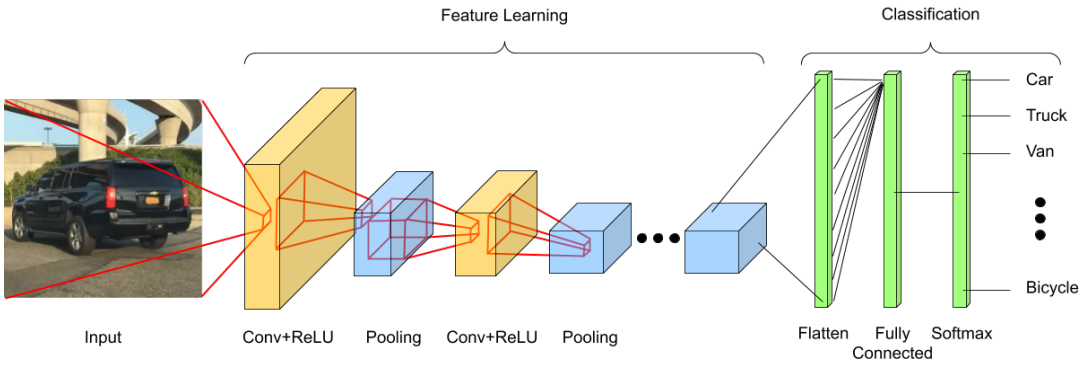

FIGURE 1.1: CNN for vehicle classification

1.3 Object Detection

Object detection is a fundamental computer vision and image processing technique, and a prerequisite of many useful computer vision tasks, such as automatic image annotation, face detection and recognition, vehicle number plate detection and recognition, and so on. Algorithms that execute object detection tasks are usually referred to as object detectors. Each object detector is designed or trained for detecting one or multiple classes of objects; Every object class has its unique set of features. A example of an object detector's typical function is to return the location of vehicles from an image or a video sequence, as shown in Fig. 1.2. With the help of deep learning neural networks, nowadays, some of the best object detectors can even reach a higher accuracy than well-trained humans, such as the method proposed by [64, 18]

in ILSVRC[3].

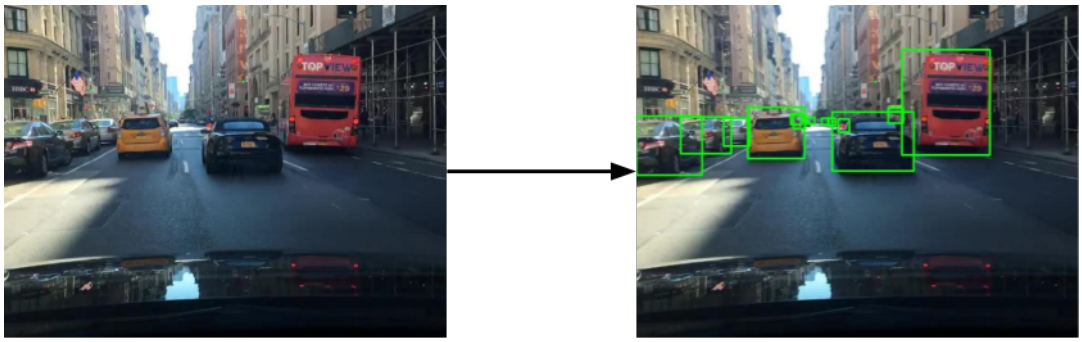

FIGURE 1.2: A Typical Example of Object Detection Task

Though object detection has come a long way, there are still many challenges to solve. For example, how to increase the accuracy when the objects are relatively small, *e.g.* cars and pedestrians shot from cameras mounted on *Unmanned Aerial Vehicles (UAV)*, which might only have few pixels. How to find a perfect balance between speed and accuracy? How to decrease the model size and makes it fit into embedded systems?

1.4 Forward-leading Vehicle Detection

When a car is driving on the road, vehicles in front of the car are considered as leading vehicles. [16] proposed a leading vehicle localization method by combining lane detection, leading vehicle detection and tracking together. It reaches over 90% precision and recall but can only run 10 frame per second on a 3.6GHz Intel i7 processor.

In this thesis, we focus on three specific cases of leading vehicles for building our FCW system: leading vehicles in the same lane, merging into the lane, and departing from the lane. We define the closest leading vehicle in those three cases as the forward-leading vehicle. In our FCW system, the forward-leading vehicle detection module contains two parts. The first part is to detect all the visible vehicles in an image, while the second part being identifying which car is forward-leading and which ones are not.

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

1.5 Problem Statement

Using sophisticated ADAS functions, such as *Forward Collision Warning (FCW)*, could be important for transportation companies that have a large fleet of vehicles. Replacing the current vehicles with new ones or upgrading them with cutting-edge sensors (*e.g.* LIDAR) would be time-consuming with a heavy financial burden on companies. We need to build an FCW system using just a basic sensor and a low-cost compute unit. In our case, each car has a smartphone installed on the inside of the front windshield, with its primary camera (a monocular camera) facing the front. That camera is the only sensor available to us.

The project was motivated by research collaboration with a truck fleet management company, Viewnyx, that offers in-vehicle driver assistance and monitoring features using a smart dash cam.

1.6 Contributions

In this work, we propose a forward leading vehicle collision alert system operating on smartphones.

The major contribution of our work includes:

- We analyzed the shape distribution of vehicles in two datasets (one public, one collected) to customize object detection models.
- We customized popular object detectors by generating more efficient default boxes. Compared with the original detectors, our most successful method uses 38.4% fewer default boxes, provides better performance, and slightly higher frame rate, as well as lower memory/battery requirement.
- We tested multiple object tracking techniques and used a detection-tracking scheme to boost leading vehicle detection. We found out that the leading vehicle detection is 3 to 5 times faster than pure detection, with a negligible performance loss.
- We propose a simple monocular distance estimation method that is suitable for smartphone cameras.
- With the fast leading vehicle detector and monocular distance estimation, we designed a forward leading vehicle collision alert system operating on Android smartphones. We also provide a Python version of this system.

1.6.1 Structure of the Thesis

The article is organized according to the structure of the system. The overview of each chapter is as follows:

- **Chapter 2** "Related Works:" This chapter presents fundamental knowledge about Data Preprocessing and basic concepts of CNN, along with a literature review about state-of-the-art CNN architectures, object detection models, object tracking techniques, and camera-based distance estimation methods.
- **Chapter 3** "Dataset Analysis and Architecture Selection:" In this chapter we show how to annotate image data, then some observations and analysis about datasets, such as bounding box distribution, are presented. Then we select a suitable object detector to work with for our vehicle detection task.
- **Chapter 4** "SSD Customization for Vehicle Detection:" In this chapter we propose methods to customize and optimize object detection models for vehicle detection by changing feature map sizes and obtaining optimized default boxes of SSD object detector.
- **Chapter 5** "Experimental Results of Vehicle Detection:" This chapter presents the experimental results of all the models mentioned in Chapter 4. We compare the results and select the most successful model for further developments.
- **Chapter 6** "Forward-leading Vehicle Detection Acceleration:" Using the trained models in Chapter 4 & 5, we accelerate forward-leading vehicle detection with tracking techniques.
- **Chapter 7** "Forward Collision Warning System:" We propose a simple and practical distance estimation method that only requires a monocular camera. Then, we design a Forward Collision System and implement it onto Python and Android platforms.
- **Chapter 8** "Conclusion:" Lastly, we draw the conclusion of our work and discuss future works.

Chapter 2

Related Works

Our work is about building an FCW system for low-power devices (*e.g.* mobile devices), based on forward-leading vehicle detection. The whole system contains a light-weight object detector and other techniques, such as tracking and distance estimation. In this chapter, we briefly discuss the existing problems in vision-based FCW systems, as well as the pros and cons of the existing solutions, including CNN backbones and object detection models. This chapter also provides some fundamental knowledge for understanding our works, such as basic concepts of CNN, neural network training techniques, and evaluation metrics of object detection.

The order of related works is arranged according to the entire FCW system's workflow, namely data preprocessing and analysis, object detection, object tracking, and distance estimation. Since CNN backbones are used in object detection models, we discuss CNN ahead of the object detection section.

2.1 Basic Concepts of CNN

Convolutional Neural Network (CNN) is a typical artificial network used for computing customized outputs from various inputs. In this section, we introduce the main concepts over which a CNN architecture can be built.

2.1.1 Convolutional Layers

In a CNN, convolution is applied to two matrices, the input and a smaller matrix named as the convolutional kernel. In deep learning, kernels are usually built from several 2D convolutional filters that have the same shape. In a 2D convolution, a filter is sliding through the input matrix. The output is the sum of the multiplication result of each filter cell and the input cell covered by it. An example is shown in Fig.2.1, in which the kernel only has one filter. The navy blue cells denote the centers of all the kernels, which is also the output shape. Generally, four factors determine the shape of convolution output: *stride*, *depth*, *padding*, and the kernel's shape. Depth means how many 2D filters does a 3D kernel has. Stride denotes the

number of pixels to skip when sliding the kernel through an input map. The kernel width and height are all 3 units in the figure, but it could be changed to customize specific output shapes.

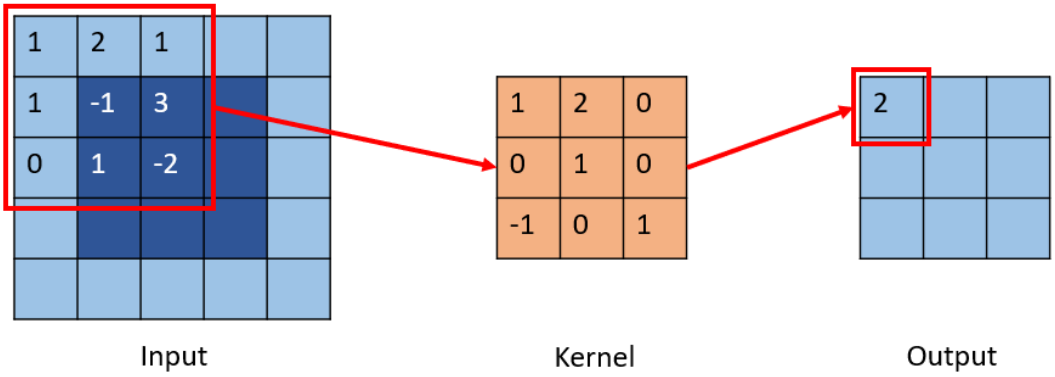

FIGURE 2.1: An example of 2D convolution.

Usually, we tend to lose pixels when applying convolutional layers. The larger the kernel is, the more pixels we lose. In Fig.2.1, the output matrix's width and height are 2 pixels smaller than the input matrix. In many cases, we want to let the output matrix and input matrix have the same shape. A technique to make it happen is *Padding*, which is extending the shape of the input matrix. If the kernel's width is k_w, to make sure the output and input matrices have the width, we can add p_w columns to both the left and right side of the input, where $p_w = \frac{k_w - 1}{2}$. Likewise, we can add rows to the top and bottom of the input with similar approach to extend the height. Suppose we have a 5×5 input, a 3×3 kernel, and use 1 as the stride, we can extend the input to 7×7 to get a 5×5 output.

2.1.2 Fully Connected Layer

As mentioned in Fig. 1.1, near the end of CNN there are often one or more FC layers. In this layer, each output neuron is connected with every input neuron in the former layer. An example is shown in Fig. 2.2.

When solving multi-class problems, the FC layer's output contains as many neurons as the number of the classes, *i.e.* one real number per class. The Softmax function is for transforming outputs in the FC layer into probabilities of classification. The following equation is the Softmax function.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}}, j = 1, ..., K \tag{2.1}$$

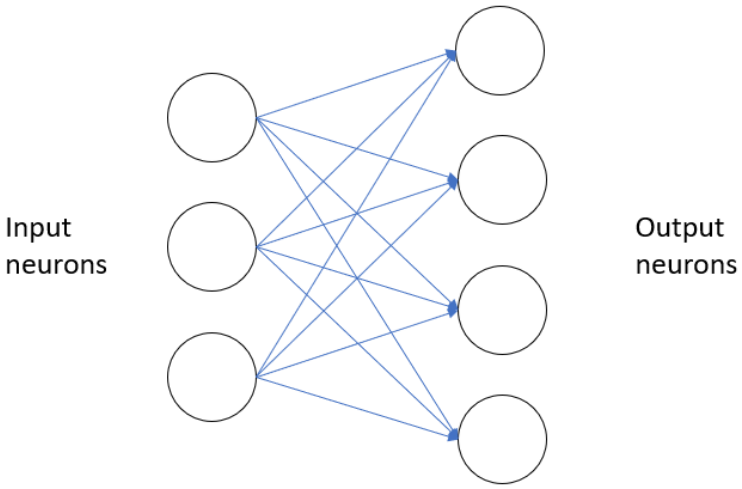

FIGURE 2.2: Fully Connected Layer.

2.2 Different CNN Backbones

There are various kinds of CNN backbones, *i.e.* the base network used in object detectors. Since our system is built for mobile devices, we mainly introduce a light-weight network designed for mobile devices, the MobileNet-V1 [26]. Our work is based on the second version of it, which is usually referred to as MobileNet-V2 [54].

To get a better understanding of MobileNet-V2, we also introduce some backbones that are related to it. We first introduce the widely used VGG-Net [59], which proves that increasing the CNN's depth could lead to higher accuracy. ResNet [19] is proposed to solve the vanishing gradient problem in training very deep neural networks. Then, we introduce the MobileNet-V1 and its core idea–the depthwise and pointwise convolution. After that, we show the refinements in MobileNet-V2 that were inspired by ResNet. In this section, we briefly explain how these backbones work, as well as presenting their performance.

2.2.1 VGG-Net

VGG-Net [59] is a network proposed during the ILSVRC-2014 competition for image classification tasks. Unlike the previous successful networks, such as AlexNet [36], that use large kernel size and receptive field, VGG-Net repeatedly uses 3×3 kernels to build a deeper network to get higher accuracy.

There are many variants of VGG-Net that have a different number of convolutional layers, *e.g.* the winner of ILSVRC-2014, VGG-16, achieves 6.8% top-5 test error with 138 million trainable parameters. The structure of the VGG-16 model is shown

in Fig. 2.3. In the figure, ReLU stands for rectified linear unit function (Ramp function).

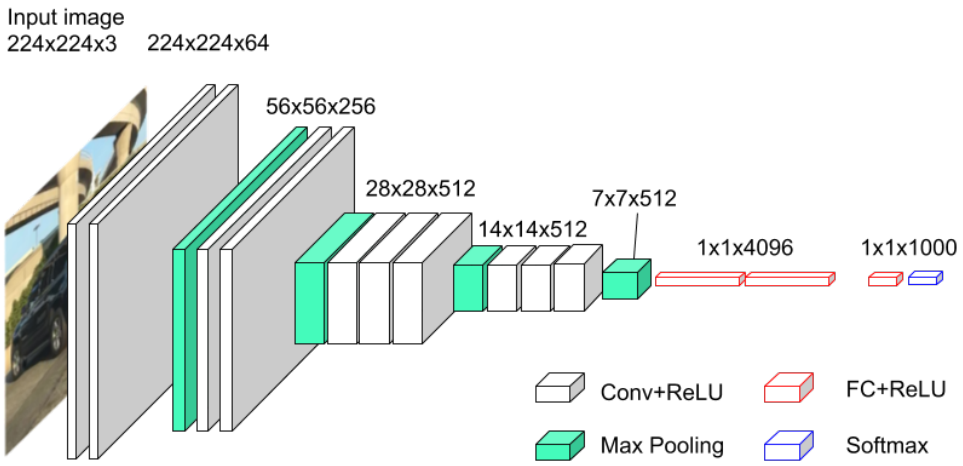

FIGURE 2.3: The structure of VGG-16.

The major limitation of VGG-Net model is that the depth of the model, *i.e.* the number of convolutional layers, cannot be too large because it usually leads to very long training time and vanishing gradient problem.

2.2.2 ResNet

As mentioned in the VGG-Net section, when a neural network is deeper, it could be harder to get trained due to the vanishing gradient problem. Here we briefly explain what is the vanishing gradient problem. Many networks are getting trained with gradient-based learning methods. These gradient vectors often have small magnitudes and are usually updated with backpropagation. In this case, the gradients are computed with the chain rule, which leads to multiplying several small numbers in a chain, such that the gradients to the first few layers are too small to update the weights, and the network will not get trained.

In ResNet [19], a residual block is proposed. An example of a residual block is shown in Fig. 2.4. Each block contains several convolutional layers. The residual identity mapping (including a 1×1 Convolution followed by batch normalization) is applied to pass the gradient between adjacent blocks, such that the error could be passed from the last to the first convolutional layer of the network.

2.2.3 MobileNet-V1

For a long time, CNN models have a large number of parameters leading to a huge number of multiply-accumulate (MACs) operations. Therefore, running CNN on

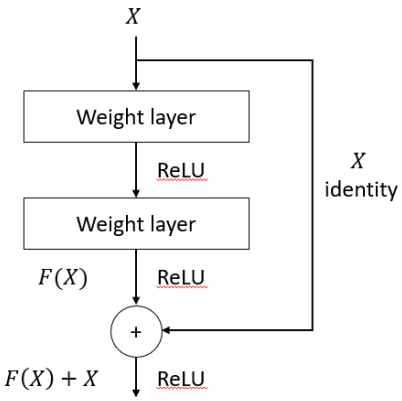

FIGURE 2.4: A residual block with two convolutional layers.

mobile devices and achieving real-time frame rate has always been a challenge. MobileNet-V1 [26] is designed for mobile devices, which replaces the standard convolution with a set of separable convolution, including *depthwise* and *pointwise* convolution, to reduce the overall computation. As a result, 95% of computing time is spent on 1×1 convolution. Fig. 2.5 illustrates the difference between standard convolution (the first row) and separable convolution (the second row). Let $D_F \times D_F$ be the input feature map's size, M and N be the depth of the input and output channels, and $D_K \times D_K$ be the kernel size (here $D_K = 3$). Therefore, the standard convolution has $D_K^2 \times M \times N = 864$ weights, while the depthwise convolution in MobileNet-V1 has $D_K^2 \times M = 27$ weights, and the pointwise convolution has $M \times N = 96$ weights. Therefore, the MACs of standard convolution is $864D_F^2$, and the MACs of separable convolution is $123D_F^2$.

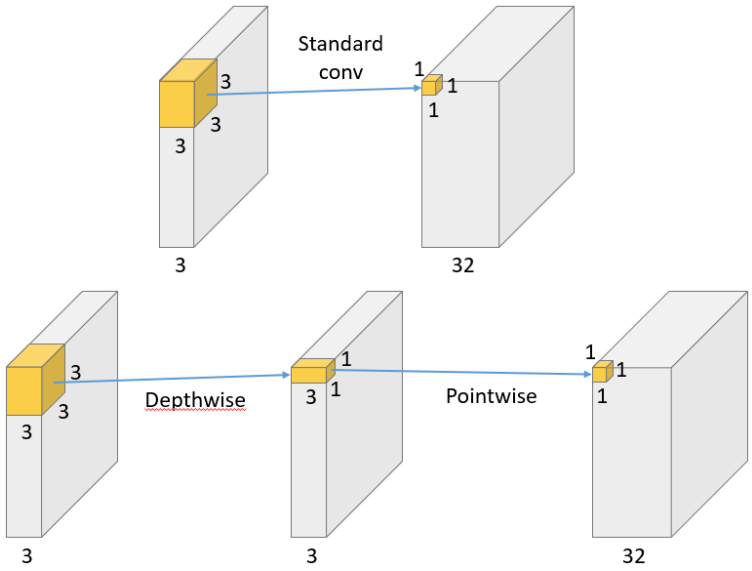

FIGURE 2.5: Standard convolution VS. separable convolution in MobileNet-V1

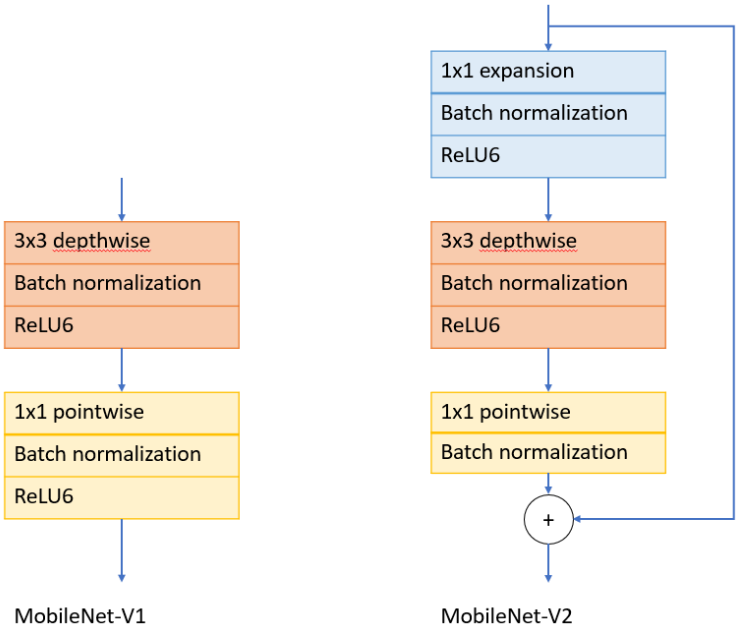

FIGURE 2.6: Basic block in MobileNet-V1 and MobileNet-V2.

2.2.4 MobileNet-V2

Inspired by ResNet, MobileNet-V2 [54] uses inverted residual blocks to make the training easier when having greater model depth. Fig. 2.6 shows the difference between the basic convolutional blocks in MobileNet-V1 and MobileNet-V2. MobileNet-V2 has fewer parameters and less computation compared with the previous version, but achieves higher performance and can be trained faster. In Section 3.3, more details about MobileNet-V2 are given, explaining why it has been selected as the backbone used in our work.

2.3 Basic Concepts of Object Detection

In this section, we introduce some basic concepts in object detection and some techniques used for training and evaluation.

2.3.1 Region Proposals

In the early object detection methods, *sliding window* was used to find objects [9, 13]. It consists of using a predefined box and sliding it through the image at all the possible positions and scales. This method guarantees that all visible objects in the image are scanned and put into an object classifier. However, this method is very computationally expensive, primarily when binding it with a deep learning object classifier to build an object detector because it implies running the classifier at all the possible locations. To alleviate this problem, R-CNN [15] was proposed. In R-CNN, around

$2k$ regions are extracted from each image, and the CNN classifier is only executed for these regions. These regions are referred as *Region Proposals*.

In R-CNN, the region proposals are acquired through Selective Search [63], which is based on hierarchical Bottom-up grouping segmentation. In each hierarchical group, the segmentation results have various average sizes that can successfully cover both small and large objects. Region proposals method provide prior information to the classifier about the shape of objects. It inspired us that to analyze the shape of objects and give the results to a classifier before training it to improve the performance. Such an idea is the motivation of our works in Section 4.2 and 4.3. Note that the object shape analysis and objector training are using the same training set. No information from the validation set or testing set can be used for the shape analysis.

2.3.2 Average Precision

Average Precision (AP) is a measure proposed as an evaluation standard in Information Retrieval. Since PASCAL VOC Challenge [12], AP is also used to evaluate object detection performance and has become one of the most popular evaluation metrics. In object detection, AP is defined from *Precision*, *Recall*, and *Intersection over Union (IoU)*, and used for detection models based on *Bounding Boxes*.

Precision and recall are defined by Equation 2.2 and 2.3, in which *True Positive (TP)*, *True Negative (TN)*, *False Positive (FP)*, and *False Negative (FN)* are defined in Fig. 2.7. For object detectors, if the precision is high, the predicted objects have a higher probability of having a correct class, *i.e.* the predicted class is the same as in the ground truth. If the recall is high, fewer target objects are missed when making predictions.

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

For any object in an image, we can find a minimum box to include all the pixels of that object. Such a box is usually referred as a Bounding Box. Intersection over Union (IoU) is defined as shown in Fig. 2.8, to evaluate whether the predicted shape and location are good or not. Generally, if a pair of predicted and ground truth boxes have a large IoU (usually using thresholds $IoU > 0.5$ or $IoU > 0.75$), we consider it a decent prediction in terms of localization.

		Ground-truth	
		Positive	Negative
Prediction	Positive	True positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

FIGURE 2.7: The Confusion Matrix for object detection.

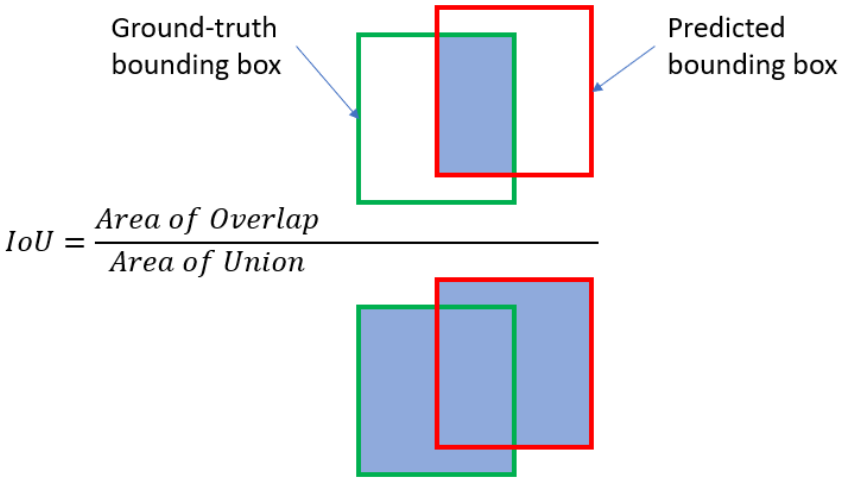

FIGURE 2.8: IoU for object detection.

AP is defined from the precision-recall curve based on Interpolated Precision [12]. The objective is to remove the wiggles on the curve caused by the small changes in the ranking of predictions. For each recall $r \in \{0.0, 0.1, ..., 1.0\}$, we can plot a precision $p(r)$. Then, AP is computed with the following equation:

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, ..., 1.0\}} p_{interp}(r) \tag{2.4}$$

where $p_{interp}(r)$ is the interpolated precision at each recall level r, defined as the maximum precision measured for which the corresponding recall is greater than r. The mathematical definition is:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{2.5}$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$. For example, if we have $p(r = 0.9) = 0.2$ and $p(r = 1.0) = 0.3$, the precision-recall curve will have a zigzag pattern.

The interpolated precision at $r = 0.9$ is the maximum of the precision values where the corresponding recall is larger than 0.9, which gives:

$$p_{interp}(r) = \max\{p(r = 0.9), p(r = 1.0)\} = p(r = 1.0) = 0.3 \tag{2.6}$$

As a result, the curve of interpolated precision p_{interp} and recall r does not have wiggles. The higher the AP is, the better the detector we have.

If there is more than one class in the detection task, the mean of the AP of each class is computed, which is referred to as *mAP*. There are also some customized versions of AP, such as the AP_{50} and AP_{75} in [39] that at certain IoU level.

2.3.3 Non-Maximum-Suppression

Usually, in object detection tasks, the number of raw predictions is large. Hence, there are often more than one predictions near a ground truth box. *Non-maximum Suppression (NMS)* is a technique for reducing redundant predictions. The input of NMS includes a list of predictions L_{in}, as well as the corresponding IoU and confidence scores. The method is initialized with an empty output list named L_{out}. From the input list L_{in}, the box with the highest confidence score, referred to as B, is selected and moved into the output list L_{out}. Then B is compared with the rest of boxes; if the IoU between a box and B is smaller than a threshold T_{iou}, that box is removed from L_{in}. These two steps are repeated until the L_{in} is empty. This way, around each object, only the bounding box that has the highest confidence score and decent IoU is kept in L_{out}, most of the redundant boxes are removed. An example is shown in Fig. 2.9.

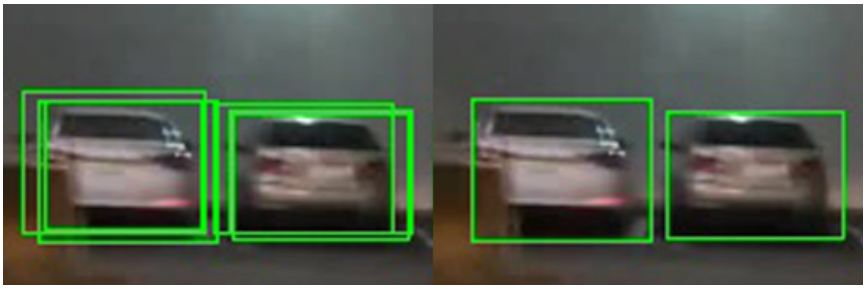

FIGURE 2.9: Prediction results before and after NMS.
Left: before . Right: after.

There are many variants of NMS, such as the traditional methods mentioned in [45], and CNN-based NMS [25]. In our work, our object detection models are built with TensorFlow [1]. Instead of using the built-in NMS functions in TensorFlow, we implement NMS in Python and apply it to our models. This is because the NMS

function in TensorFlow is using a tensor-like structure to save the data, and including many other extra steps, such as calculating and recording the gradient, which is not needed for our work. NMS processing requires less computing time and less memory with such a tweaking compared with the TensorFlow NMS. In Section 5.3.4, we discuss post-processing time in object detection models that is mainly caused by NMS.

2.4 Object Detection Models

In the last few years, object detection has been a fast-evolving field where a large number of CNN-based object detection models have been proposed. There are mainly two types of object detection models. The *two-stage* object detectors are based on region proposal. The main ones are Fast RCNN [14], Faster R-CNN [51], and Mask R-CNN [20] (although the latter is designed for pixel-level image segmentation, it could be used for detection). In such models, the first stage proposes numerous regions by applying *Select Search* or a *Regional Proposal Network (RPN)*. The second stage is throwing the proposed regions into a classifier. *One-stage* detectors do not propose regions but use designed or randomly generated dense sampling regions and run detection on all of these regions. The popular one-stage detectors include YOLO [50], SSD [41], and RetinaNet [38], as well as the many variants of these models.

Generally, two-stage detectors are "heavier" than one-stage detectors, which means they have more trainable parameters, as well as more multiply-accumulate operations. These lead to a higher memory requirement and computing capability. Consequently, being heavier means better mAP but a much lower frame rate. In our work, pre-trained two-stage object detectors are used for dataset preprocessing because, with their high mAP, they can reduce the workload of annotating. Only one-stage detectors are considered in our work because they are "light" enough to be implemented on mobile devices.

Many new object detection models are proposed after completing our work, such as YOLO-V4 [5] and EfficientDet [61]. Building FCW systems with these state-of-the-art object detectors might become a part of our future work.

2.4.1 Faster R-CNN

Faster R-CNN [51] is one of the most successful successor in the R-CNN [15] family. It is also the main model used for producing annotations of our collected dataset. It includes three major parts: an RPN for finding objects on images, a CNN that takes

original size images as input and creates feature maps, and an *Region of Interest (RoI)* feature vector that passes the feature map in that region to a classifier. The second and the third part are also known as Fast R-CNN [14] classifier.

There is a fundamental concept in Faster R-CNN named *Anchor Boxes*, which is a set of hand-picked boxes with different aspect ratios that are used for RPN. At each sliding-window location of the RPN, multiple region proposals are predicted based on the anchor boxes' scale and shape. An example is shown in Fig. 2.10. Such design is also used in many other models.

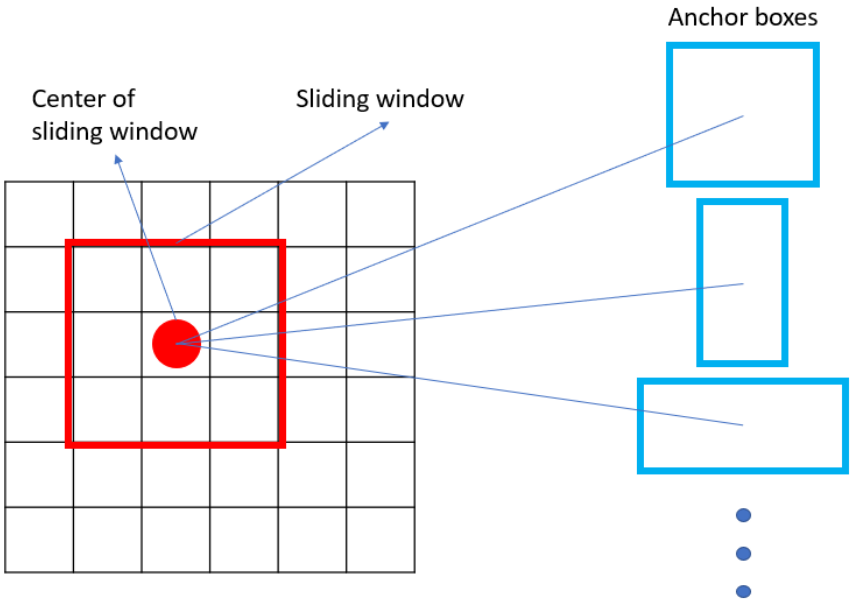

FIGURE 2.10: Anchor boxes for RPN.

Faster R-CNN is considered as one of the most powerful object detectors that can get very high precision. The original Faster R-CNN reaches 78.8% mAP on the COCO [39] dataset with only 300 proposals per image. However, it can only run at few frame per second (FPS), even on a powerful GPU, due to its high computational cost. Therefore, Faster R-CNN is usually used for mAP-oriented tasks only. In our work, this model is used for producing annotations in our collected dataset.

2.4.2 YOLO-V1

In 2016, Redmon et al. proposed the first one-stage object detector that can run at real-time, known as YOLO [50]. It splits the whole input image into a $S \times S$ grid, with each cell in the grid being responsible for predicting bounding boxes and the boxes' confidence score. The confidence score shows how confident the model is that an object is included in that box, as well as whether the box is highly overlapping

with the ground truth box. Therefore, for each class of objects, each prediction includes the box location and confidence. This work's base model reaches 45 FPS and 63.4 mAP on Pascal VOC 2007 [11] dataset, while the smaller version of this model reaches 155 FPS and achieve 52.7 mAP. This performance is considerably higher than the result of other real-time object detectors such as DPM [53], and even R-CNN. The major shortcoming of this model resides in the difficulty of detecting small objects that are close to each other.

2.4.3 YOLO-V2 and YOLO-V3

YOLO-V2 [48] is an improved YOLO model, which uses higher input resolution and apply anchor boxes for box localization prediction. In each grid cell, a set of anchor boxes is used to let the network know the shape of the objects. These anchor boxes used to be designed manually in Faster R-CNN. In YOLO-V2, it takes the boxes in the training set, applies K-Means clustering to the boxes width and height, then takes the centroids as the anchor boxes. Some other tricks, such as *batch normalization*, are also applied to YOLO-V2 to improve the mAP.

YOLO-V3 [49] is a technical report after YOLO-V2, which provides more tweaks and tricks to push the performance of YOLO-V2. For example, it uses *shortcut connections* to pass the feature in the first few layers to the end of the network, such that details of small objects are not smoothed out during pooling, which improves the mAP on small objects. Eventually, YOLO-V3 reach 55.3% mAP and 34.5 FPS on the COCO dataset [39].

2.4.4 SSD

The Single Shot Detector (SSD) [41] model uses a pyramidal feature hierarchy for efficiently detecting objects of various sizes. It uses the VGG-16 model in [59] as the backbone feature extractor. The architecture of this SSD VGG-16 detector is shown in Fig. 2.11. SSD is the primary object detector that is used and customized in our work. We explain two parts in SSD: multi-scale feature maps and default boxes, as our works are highly related to them.

Multi-scale Feature Maps

There are six feature layers in SSD, which are referred to as *Multi-scale Feature Maps* in the paper of SSD [41]. The first SSD feature map is derived from a layer in the middle of VGG-16, while the second SSD feature map is derived from the last layer of VGG-16 through convolution. The third to the sixth SSD feature maps are derived from their previous SSD feature map through convolution. Let the shape of the l^{th}

layer of feature map be $w_l \times h_l \times d_l$, where the w_l, h_l, and d_l denote width, height and depth. Usually, we have $w_l = h_l$.

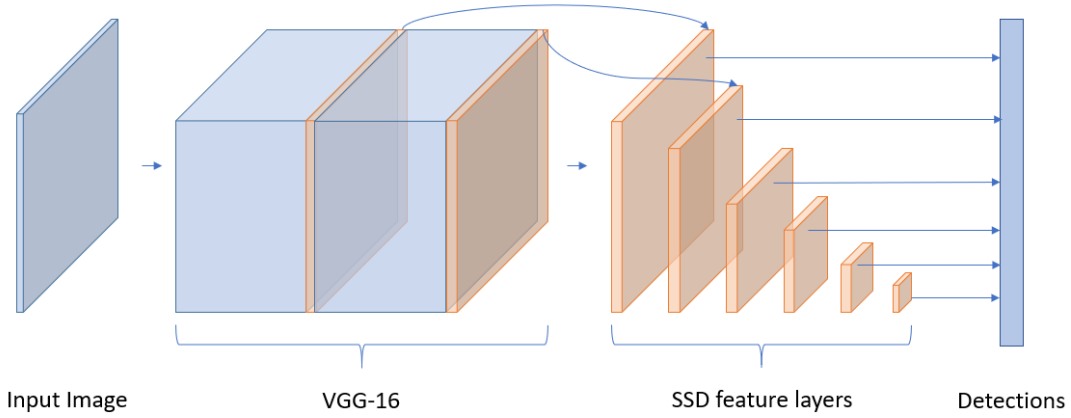

FIGURE 2.11: Model architecture of SSD.

Default Boxes

For an SSD feature layer whose width and height are w_l and h_l, we consider it has $w_l \times h_l$ cells. Each cell is responsible for predicting the location and the confidence scores of objects. The location includes the width w, height h of the objects, as well as the x-axis and y-axis offsets, cx and cy. c_p denotes confidence score the p^{th} class in a classification task.

SSD uses pre-defined *Default Boxes* for making such predictions, which is similar to the anchor boxes used in the Faster R-CNN and models in the YOLO family. The default boxes tile the feature map in a convolutional manner, so that we have a set of default boxes for each cell. A default box can be described with *baseline size* s and *aspect ratio* a, where a equals to the box width divided by box height. The box width and height are integers. For example, Fig. 2.12 shows two default boxes. The one plotted with solid lines has $s = 10$ and $a = 1$, which makes it a $s \times s$ square. The other has $s = 10$ and $a = 2$, whose box width is $w_b = s \times \sqrt{a} \approx 14$ and box height is $h_b = \frac{s}{\sqrt{a}} \approx 7$.

The baseline sizes for the l^{th} layer in the original SSD is $s_l = r_l \times I$, where I is the input size, and r_l is a scalar defined by the following equation.

$$r_l = r_{min} + \frac{r_{max} - r_{min}}{m - 1}(l - 1), l \in [1, m] \tag{2.7}$$

where $r_{min} = 0.2$, $r_{max} = 0.9$, m is the number of layer, l is the index of a layer. For example, if the input size is 150×150 ($I = 150$) and there are six multi-scale feature maps, we have $r_l \in \{0.2, 0.34, 0.48, 0.62, 0.76, 0.9\}$. Such set of r_l gives

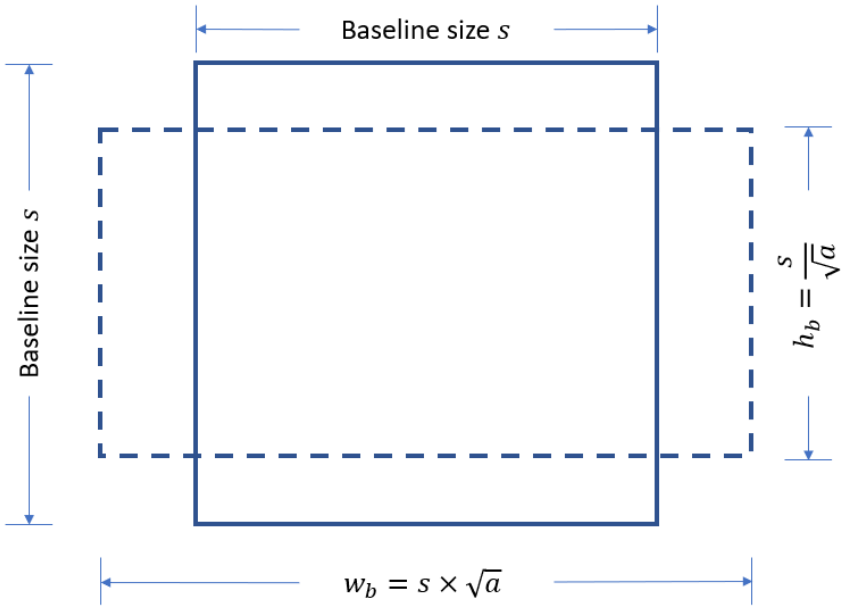

FIGURE 2.12: A square and a rectangular default boxes.

the baseline sizes $s_l \in \{30, 51, 72, 93, 114, 135\}$. In the original SSD model, it uses $a \in \{1.0, 2.0, 0.5, 3.0, 0.333\}$ for all the multi-scale feature maps to generate five default boxes for l^{th} layer. Then, one additional default box is created with $a = 1$ and a new scalar $s'_l = \sqrt{s_l s_{l+1}}$ and appended to the l^{th} layer. As a result, at each location of a feature map, there are six default boxes.

An example of a 5×5 SSD feature map and a set of default boxes are shown in Fig. 2.13. There is a prediction for each default box in each feature map cell, including classification and localization. Therefore, the number of default boxes determines the total number of predictions. For example, in Fig. 2.13, we have four default boxes for each cell, and the layer width and height are $w_l = h_l = 5$. Therefore, for this layer, we have $4 \times w_l \times h_l$ predictions in total.

Loss Function

In SSD, the overall loss is computed by a weighted sum of localization loss L_{loc} and confidence loss L_{conf}, shown as:

$$L(x, c, l, g) = \frac{1}{N}(L_{loc}(x, c) + \alpha L_{loc}(x, l, g)) \tag{2.8}$$

where N is the number of the matched default boxes. The loss is set to 0 if $N = 0$.

In our work, we only have one class. Let $x_{ij} = \{1, 0\}$ be an indicator that shows whether it is a successful match between the i^{th} default box and the j^{th} ground-truth

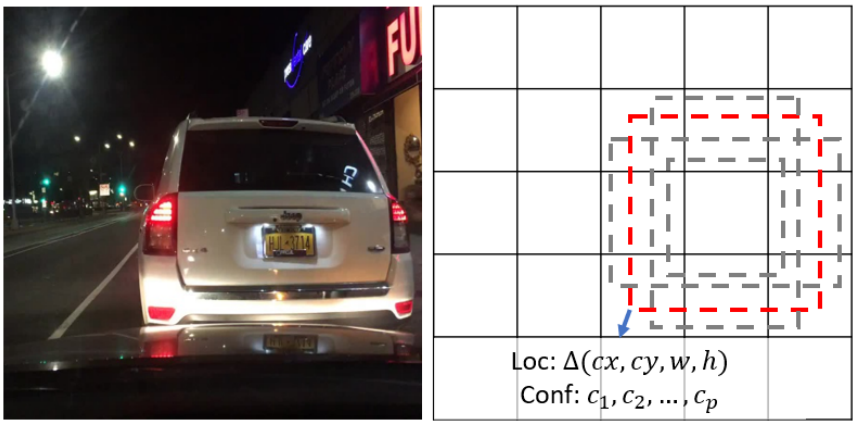

FIGURE 2.13: Default boxes of a cell in a SSD feature map.
The left figure is a resized input image. The right figure shows a set of default boxes of a cell, which are set to have different aspect ratios and scales to fit the SSD feature layers.

box. If the IoU between a default box and a ground-truth box is greater than 0.5, we consider it a successful match. The localization loss $L_{loc}(x, l, g)$ is a Smooth L1 loss [14] between ground truth boxes g and predictions l, shown as followed.

$$
\begin{aligned}
L_{loc}(x, l, g) &= \sum_{i}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij} Smooth_{L1}(l^m - \hat{g}_j^m) \\
\hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx}) / d_i^w \\
\hat{g}_j^{cy} &= (g_j^{cy} - d_i^{cy}) / d_i^h \\
\hat{g}_j^w &= log(\frac{g_j^w}{d_i^w}) \\
\hat{g}_j^h &= log(\frac{g_j^h}{d_i^h})
\end{aligned}
\tag{2.9}
$$

where d denotes default boxes, N is the number of matched default boxes. During training, reducing the localization loss could be considered as regressing default boxes to nearby ground-truth boxes.

In the original SSD paper, the confidence loss $L_{loc}(x, c)$ is the softmax loss over multiple classes confidences c. However, in our thesis, we only have one class, which gives:

$$
\begin{aligned}
L_{loc}(x,c) &= -\sum_{i\in Pos}^{N} x_{ij} log(\hat{c}_i) - \sum_{i\in Neg} log(\hat{c}_i^0) \\
\hat{c}_i &= \frac{exp(c_i)}{\sum exp(c_i)}
\end{aligned}
\tag{2.10}
$$

Performance

For a 300×300 input, SSD can run at 59 FPS on Titan GPU and achieve 74.3% mAP on Pascal VOC 2007 dataset. SSD is famous for being fast and efficient in detecting medium-size and large objects, but ineffective on small objects. Many techniques have been applied to the SSD to improve the mAP, such as inserting interval layers to the SSD feature layers and increasing the number of default boxes. One of our major research is about how to generate fewer but more efficient default boxes, which will be introduced in Section 4.2 and 4.3.

2.5 Object Tracking

Object tracking is the task of locating a moving object in consecutive camera frames. There are several successful off-the-shelf object trackers that are based on *Correlation Filters*, such as MOSSE [6], CSRT [42], and KCF [23, 22] trackers. The MIL [4] tracker uses multiple instances per training and gets a single classification result. The TLD [32] tracker decomposes the tracking task into three parts, namely tracking, learning, and detection. The MedianFlow [31] tracker evaluates both the forward and backward errors in a timed sequence to perform tracking.

In our work, we attempt to boost our FCW system by replacing some detection phase with tracking phase because our CNN-based detector is relatively slow, while many object trackers are simple and fast. The object trackers mentioned above will be applied to our collected dataset for a vehicle tracking task. After a simple IoU comparison, the best-performing one will be used in our FCW system. Some CNN-based object trackers are also available, such as GOTURN [21]. They are out of our consideration because of their high computation. More experiments about object trackers are provided in Section 6.4.

2.6 Distance Estimation Based on Monocular Camera

Distance Estimation is an essential part of an ADAS system and especially for building FCW systems. Many kinds of sensors have been used in such a task to achieve

high accuracy, such as LIDAR, Radar, and mono/stereo cameras. However, these sensors are often expensive, economically and computationally. In our work, an economical approach is proposed, which implements an FCW system on smartphones. Therefore, the only sensor available is a monocular color camera.

Estimating distance of objects with a monocular camera is challenging because we cannot extract depth or distance information directly from the data itself, *i.e.* RGB images. Many studies about mono-camera distance estimation have been done, which include two major types: depth estimation for the entire image, and object distance estimation. Though only object distance estimation is used in our work, both of these two topics will be covered in this section because some of the object distance estimation works are based on depth estimation.

2.6.1 Depth Estimation

Unlike using a stereo camera, depth estimation with a monocular camera cannot use disparity estimation to compute pixels' depth, making it a very challenging task. Given such difficulty, monocular cameras are usually used as a complementary part of complex depth estimation methods. For example, Weifeng Chen *et al.* [7] shows a dataset name "Depth in the wild" which is annotated with relative depths between each pair of random points. It trains a pixel-wise depth prediction network with RGB-D data and relative depths, such that the network can estimate the depth of the input images through a novel loss function based on relative depth.

Joglekar *et al.* [30] apply image geometry together with the monocular camera parameters to estimate the depth. This method is beneficial for places where the geometry could be calculated, such as a road scene with driving lanes. The major shortcoming with this method is that the performance depends highly on the input resolution, especially when calculating depth for larger distances.

2.6.2 Object Distance Estimation

In our FCW system, we need object distance estimation to inform the driver about the distance to the forward leading vehicle after object detection.

Usually, vehicle distance estimation methods are designed based on the geometrical relations between the vehicle and the monocular camera. In [47], the leading vehicle's bottom line in images is used to calculate the distance. The bottom line could be obtained in many ways, such as the bottom of the bounding boxes in object detection. Such a method is used in [2] with a Haar-like vehicle detector. These methods are fast and relies on camera calibration. In our work, an accurate distance

estimation method is proposed (see Section 7.2), which is also based on camera calibration.

There are also CNN-based distance estimation methods, such as DistanceNet [35] and DisNet [17]. Compared with methods based on camera calibration, these CNN-based methods provide considerably lower estimation error, but possess larger computation and require larger memory for the hardware. In our work, CNN-based distance estimators are not considered because of two reasons. First, our object detection is already CNN-based and our hardware only has limited computation resources. Second, training such distance estimators requires an accurately annotated dataset. However, the ground truth distance in our collected dataset is too noisy. Nonetheless, one of our future work could be fusing the leading vehicle detection and the distance estimation into a single network to achieve better performance.

Chapter 3

Dataset Analysis and Architecture Selection

We now live in a data-driven society, and data is considered as a gold mine. It is in this context that data mining and data analysis has become particularly important. As mentioned in Chapter 1, a part of this thesis is about vehicle detection, which consists of predicting vehicles' position with images as input. However, before making such predictions, we need to transform the raw images into machine-readable formats, which is to perform what we call *Dataset Preprocessing*. This technique could involve several methods, such as annotation, data encoding, and data validation. To train vehicle detectors, we need to find the layout of vehicles on the images first, *i.e. Region of Interest (ROI)*, then give them a label, *i.e.* the class of the object. This process is referred to as *Annotation*.

After dataset analysis, we introduce some experiments among several state-of-the-art object detectors, which help us choose an architecture suitable for vehicle detection task and our mobile device.

The chapter structure is as followed. In Section 3.1, we show how to produce annotations for our collected dataset using bounding boxes. Then, in Section 3.2, we present some interesting observations about the bounding boxes of vehicles in our collected dataset and in two publicly available datasets. The observation stimulated our next works about customizing bounding boxes' size and aspect ratio. In Section 3.3, we explain why we choose the SSD MobileNet-V2 as our base model.

3.1 Annotating Objects

In object detection, several different annotation formats are used. Some commonly used annotation formats are given in Fig. 3.1.

For vehicle detection, we have 3D and 2D detectors. 3D detectors usually have high computation and can predict the 3D position and orientation of vehicles. 2D

FIGURE 3.1: Different types of annotations.
From the left to right side of the figure, the annotation formats are *Bounding Boxes*, *Polygons*, *Pixelwise Mask*, *Cuboids* and *Keypoints*.

detectors usually take considerably fewer computing resources than 3D detectors, providing fewer dimensions or less accurate predictions. Since our targeted processing unit is a smartphone, training a 3D Cuboids-based detector is not an option because of the high computational cost it involves. A more practical way is to design a 2D object detector. We choose to use Bounding Boxes as our annotation format. Each bounding box defines a minimum box that contains all the pixels of a vehicle. In our work, annotations are expressed as 1×4 arrays of $[x, y, w, h]$, where x and y are the coordinates of the top left corner of the bounding boxes, w and h are the width and height of the bounding boxes.

Each bounding box is associated with labels. In this thesis, the first label corresponds to the vehicle type *car*, *truck*, and *bus*. The second label indicates whether or not a vehicle is forward-leading. We use *leading* for forward-leading vehicles and *sideways* for the others. Since we focus on leading vehicles only, the vehicles in the opposite direction are also included in the sideways category.

The images in our collected dataset were taken from a camera mounted on the visor of trucks, of which the image resolution is 640×480 pixels. We annotate all the vehicles using a pre-trained Faster-RCNN object detector and then correct the annotation manually. An example of annotated images is shown in Fig. 3.2. This collected dataset will be referred as *Viewnyx dataset* in the rest of this thesis.

3.2 Bounding Box Distribution

This section presents some analysis of bounding boxes distribution with respect to width, height, diagonal, and aspect ratio. We used two publicly available datasets, which are BDD100K [65] and Caltech Pedestrian Dataset [10] (designated as Caltech dataset in the rest part of this thesis), as well as our manually annotated dataset, the Viewnyx dataset.

FIGURE 3.2: Example of an annotated image in Viewnyx dataset.

3.2.1 Dataset Introduction

BDD100K is a dataset developed by the Berkeley Deep Drive team and it has 100,000 images (in). It provides a wide range of images about daily road scenes for many computer vision tasks, such as road object detection, instance segmentation, identifying driving areas and lanes. For object detection tasks, it provides many classes (*e.g.* car, truck, bus, train, bike, person, etc.) in the form of Bounding Boxes and Pixelwise masks. In our case, we used the bounding boxes annotations and only consider the vehicles category, *i.e.* car, truck, and bus. During the training process, all the images in this dataset have been cropped to 4:3 from the image center (*i.e.* from 1280×720 to 960×720 pixels), then resized to 640×480 pixels.

Caltech dataset has roughly 250,000 video frames with a total of 350,000 bounding boxes of pedestrians. This dataset is only used in this chapter, to discuss why bounding boxes distribution could be used as prior information to train object detectors.

The Viewnyx dataset has been produced from data captured by the Viewnyx Corporation [1]. The dataset has over 5000 images and 23 videos sequences, which contains roughly 55,000 vehicles. We applied a pre-trained Faster-RCNN object detector to the images to get rough predictions of vehicles and then correct the annotations manually. All the images and video frames have a resolution of 640×480 pixels. The images are shuffled and used for training and validation. The video sequences have a sampling rate of 10 frame per second, and their length varies from 9 to 28 seconds.

[1] https://www.viewnyx.com/

We picked 10 videos for training, 2 videos for validation, and 11 videos for testing. In this dataset, RADAR data is also provided along with the video sequences, which can show the distance and speed of the forward-leading vehicle. We use RADAR distance as the ground truth distance in our later work about distance estimation.

3.2.2 Bounding Boxes Distribution Analysis

This section's key insight is that the shape and size distribution of bounding boxes in a dataset can carry useful information that can be exploited in the design of an object detector module. We show in Fig. 3.3 the distribution of the width, height, diagonal for the different datasets. In this figure, the the first column graphs are related to the parent class, vehicle, in the BDD100K dataset. The second column presents the pedestrian class in the Caltech dataset. The third column is the vehicle class in the Viewnyx dataset. The three rows show the histograms of respectively the width, height, and diagonal length of the training sets' bounding boxes. In each sub-figure, the vertical axis is the number of bounding boxes, and the horizontal axis is the size in pixels. All the images in the three datasets have a resolution of 640×480 pixels. Hence, the maximum value of the horizontal axis of width, height, diagonal is $640, 480$, and 800. Note that all the histograms are related to the training set only.

These three datasets contain much more small objects compared to large objects. For most deep learning object detectors, it is more difficult to detect small objects than detecting large objects, as small objects usually carry less information, *i.e.* color details and patterns are more difficult to identify from far. In our case, large objects matter more, as the closer the vehicles to the camera, the more dangerous they could be. Therefore, we choose to remove objects whose width or height is smaller than a certain threshold. In our case, this number is 22 pixels for both width and height, which is named as *rejection size*. For more details about how we got this rejection size, refer to Appendix B. Note that the rejection size is only applied on the BDD100K dataset and Viewnyx dataset, not the Caltech dataset. We name the BDD100K dataset after small boxes removing as *filtered BDD100K*. Using the proposed threshold value, 47.88% boxes in BDD100K dataset and 48.13% boxes in Viewnyx dataset are removed.

Fig. 3.4 shows the aspect ratio (*i.e.* box width divided by box height) distribution of the ground truth boxes of the vehicle class in the filtered BDD100K dataset, Viewnyx dataset, and the pedestrian class in the Caltech dataset. In the figure, the vertical axis is the number of bounding boxes, and the horizontal axis is the aspect ratio. First, we check the lower and upper boundary of the range, then calculate their mean value. The majority of bounding boxes in the BDD100K dataset and Viewnyx dataset have similar aspect ratios, ranges from 0.0 to 3.0. The mean value of aspect

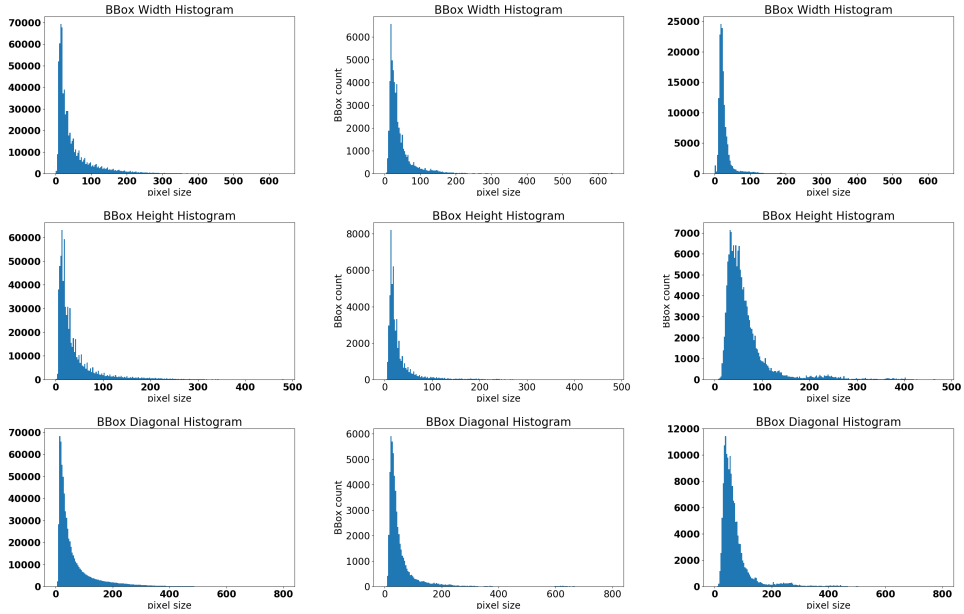

FIGURE 3.3: Width, height and diagonal distribution of Bounding Boxes in BDD100K, Viewnyx and Caltech dataset.

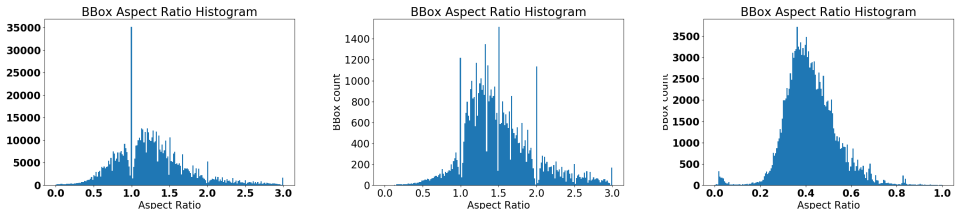

FIGURE 3.4: Aspect ratio distribution of Bounding Boxes
From left to right: Vehicle class in BDD100K and Viewnyx dataset, and pedestrian class in Caltech dataset.

ratio of BDD100K is 1.31 and Viewnyx is 1.49. These ratios are relatively close because the entities in these two datasets are all vehicles. On the contrary, the aspect ratio distribution of Caltech dataset is very diffrent from the BDD100K and Viewnyx dataset, which ranges from 0.0 to 1.0, and the mean value is 0.42. The similarity of the first two histograms and the obvious difference in the third histogram indicate that different types of objects carry significantly different bounding box aspect ratios. Hence, the aspect ratio of bounding boxes could be used as a feature of objects. If we customize an object detector based on this information, we might improve the performance.

3.2.3 Bias on Aspect Ratio of Ground Truth Boxes

In Fig. 3.4, it can be observed that there are small pulses and valleys in the aspect ratio histograms of BDD100K and Viewnyx datasets. For example, in the BDD100K histogram, there is an significant pulse at 1.0, and smaller pulses at 2.0 and 3.0. There are also valleys of different sizes around these pulses. In this section, we discuss a

possible reason for such a phenomenon.

One possible reason for this phenomenon is related to the way the datasets are made. The BDD100K and Viewnyx dataset were developed by pre-trained object detectors together with human annotators. First, images in that dataset are processed by the pre-trained object detector, and predictions of objects are proposed. After that, the predictions are corrected manually by human annotators. Generally, a well-trained object detector like Faster-RCNN has decent performance and offers excellent predictions of objects. The position errors between predictions and the actual object boundaries are often only a few pixels. However, sometimes human eyes are not sensitive enough to distinguish the difference. Hence, in a large number of cases, the human annotators leave decent predictions as they are and carry on correcting the next prediction. As a result, after manual correction, a large number of ground truth bounding boxes are the prediction results of pre-trained object detectors. If the pre-trained detector is using pre-defined anchors, this can result in a bias.

As shown in Fig. 3.4, the BDD100K dataset has many annotations around a specific set of aspect ratios, $\{1.0, 2.0, 3.0\}$. Our Viewnyx dataset was developed with a pre-trained Faster-RCNN detector using aspect ratios of $\{0.333, 0.5, 1.0, 2.0, 3.0\}$. On the aspect ratio histogram of Viewnyx dataset, there are also pulses around $\{1.0, 2.0, 3.0\}$. Therefore, we deduce that the aspect ratio bias in the BDD100K dataset is probably also created by a pre-trained Faster-RCNN detector with its anchor using a specific set of aspect ratios containing the aspect ratios $\{1.0, 2.0, 3.0\}$.

3.3 Architecture Selection for Mobile Devices

Our next step is selecting an architecture suitable for our vehicle detection task on mobile devices after dataset analysis. Since one of our main challenges is the computational cost, when we select a base model from various state-of-the-art object detectors, we give higher priority to shorter calculation time rather than higher accuracy. As mentioned in Section 2.4, single-stage object detectors usually run faster than two-stage object detectors. Therefore, we decided to choose our base model among two popular single-stage approaches, SSD[41] and YOLO[50], including their variants. We consider four factors: memory usage, model size, computational cost, and battery drain.

3.3.1 Memory and Model Size

As we aim to design an object detector for mobile devices, the saved model's file size is an important factor as it affects both the ROM and RAM usage. Usually, a saved

model contains the structure and the parameters (weights and bias) of the model. There are several ways to save a model, *e.g.* in TensorFlow [1] it uses *checkpoint* (.ckpt file) and *frozen inference graph* (.pb file). Generally, the parameters take up the largest proportion of the file size, and the larger the saved model is, the more parameters it has. In a standard workflow of running an object detector, the saved model is loaded to the memory (RAM) first. Then, the executable model of object detector is created according to the pipeline, structure, and the weight/bias data. The whole memory usage of the object detector could be considerably larger than the size of the saved model itself (usually 10 to 100 times). Since mobile devices have a limited size of RAM compared with PC, the less RAM our object detector uses, the better. Choosing small models can also reduce ROM usage. Therefore, we trained several variants of SSD and YOLO to check their saved models' size. Table 3.1 shows the result. In the table, all the YOLO models have an input size of 416×416, as set in the original paper, which gives a good balance between performance and FPS. Likewise, the SSD models in Table 3.1 have an input size of 300×300 for the same reason. As a result, the SSD MobileNet-V2 model is better than the rest of the models in the table as it has the smallest saved model size.

Model Name	Saved Model Size (KB)
YOLOv2	199155
YOLOv2-tiny	43896
YOLOv3	242195
SSD300(VGG16)	102678
SSD300(MobileNet-V1)	21973
SSD300(MobileNet-V2)	18659

TABLE 3.1: Checkpoint file size of SSD and YOLO series.

3.3.2 FPS and mAP

The exact computational cost of deep learning models is often hard to calculate. Generally, the computational cost is determined using two metrics. The first one is the *Multiply-Accumulate Operations (MACs)* needed for processing each input image/batch. The second one is *Floating Point Operations Per Second (FLOPS)*, which denotes the computational capability of a processing unit. In this thesis, we mainly use frame rate for the comparison of computational cost, with the unit being *frame per second (FPS)*. According to results on PASCAL VOC 2007 test set [48], which is shown in Table 3.2, the YOLOv2 has higher FPS than SSD300 (VGG16).

Having a high mAP is essential for object detection. From [41],[50], and [48], we know both the SSD and the YOLO models give excellent mAP that suits our needs. Therefore, in our work, the mAP is not a key factor for choosing models.

Detection Frameworks	Input Size	mAP	FPS
YOLO	448x448	63.4	45
YOLOv2	416x416	76.8	67
SSD300	300x300	74.3	46
SSD500	500x500	76.8	19

TABLE 3.2: Detection frameworks on PASCAL VOC 2007 dataset.

3.3.3 Battery Test

The last factor is battery drain speed, *i.e.* how long can a smartphone works when running an object detector from 100% to 0% battery power. We tested the original YOLOv2 model with 416×416 input size and three SSD models with different input sizes, namely 128×128, 224×224 and 300×300. We do not run the object detectors from full-battery state to empty-battery state because it is time-consuming, and we don't need to figure out the exact time of how long the battery can keep it running. We only need to know how power-consuming the models are.

We used a HUAWEI P10 Lite smartphone for the battery test, and the setups are as follows. First, we shut down all the existing applications on the phone and charge the battery to 90%. Next, we open the demo and keep it running until the battery of the phone reaches 80%, record the time consumed. Then we charge the phone to 90% battery again and prepare for the next round. After six rounds of experiments, we calculate the average battery consumption speed (percentage per minute) and switch to the next object detection model, and redo the experiments above. Note that the parameters in this experiment are not strictly controlled, such as, in the different experiments, the initial temperatures of the battery are not strictly the same. Therefore, this experiment is somewhat indicative but not a quantitative analysis of the battery consumption.

The result is shown in the Table 3.3, where the SSD128 means a SSD model with 128×128 input size, SSD224 has 224×224 input size, and SSD300 has 300×300 input size. The YOLOv2 model is an off-the-shelf model, which uses Darknet-19 [48] as backbone network. As a result, the battery consumption speed of these models ranges from 0.42 to 0.55 %/min, which means the total battery drain time ranges from 3 to 4 hours. As it can be observed, the battery drains effect increases along with the model size, but not significantly.

Model Name	Battery Consumption (%/min)
YOLOv2	0.51
SSD128 (MobileNet-V2)	0.42
SSD224 (MobileNet-V2)	0.50
SSD300 (MobileNet-V2)	0.55

TABLE 3.3: Battery Drain Effect Comparison.

3.3.4 Choosing Model

Given the analysis in the sections above, we choose the SSD MobileNet-V2 model with 300×300 input size as our base model for further developments, as it has relatively small RAM/ROM usage and low computational cost.

Chapter 4

SSD Customization for Vehicle Detection

The first part of our Forward Collision Warning system is the forward-leading vehicle detection module, which comprises two steps. The first is to detect all the vehicles that are visible in an image. The second step is identifying which car is the forward-leading one and which ones are sideways. For purposes of this chapter, we focus on the first step.

In chapter 3 we selected SSD MobileNet-V2 as our architecture. In the first version of the Single-Shot Detector (SSD) [41], the default boxes were named *priors* because both the boxes' baseline size and aspect ratio use experimental value and hence provide prior information to the model. In our work, we propose to take our dataset analysis result as prior information, then customize the multi-scale feature maps of SSD, as well as the baseline sizes and aspect ratios of the default boxes.

In Section 4.1, we propose to customize the SSD multi-scale feature maps based on the *Cumulative Distribution Functions (CDF)* of box diagonal distributions, such that each feature map layer is responsible for matching roughly the same number of ground-truth vehicles in datasets. Then, we propose two methods to customize the default boxes of SSD: Section 4.2 shows our first strategy, which is obtaining the baseline sizes and aspect ratios according to the Gaussian distribution of the ground-truth boxes. In Section 4.3, we propose our second strategy, which is applying layer-wise K-Means clustering to the ground-truth boxes to get optimized numbers, sizes, and aspect ratios of the default boxes.

4.1 Optimizing SSD Multi-scale Feature Maps

In the original SSD model, there are six multi-scale feature maps. As the first few feature maps are larger than the rest, the default boxes associated with those layers are smaller and will be responsible for matching small ground-truth boxes, *i.e.* distant vehicles. The distant vehicles are less important to us, which deserve less

computation in our task. Therefore, our first objective is to find a better balance of SSD multi-scale feature maps. Our strategy is to make each feature map account for matching roughly the same number of entities.

We keep the number of the feature maps as in the original SSD for comparison. We then focus on changing the size of the feature maps. To this end, the convolutional kernels' size is not increased but kept as 3x3 to avoid raising the computation. Besides, many off-the-shelf SSD implementations use interpolated feature map layers, which inserts an additional feature map layer between each two consecutive feature map layers. Using interpolated layers is a typical engineering technique that can increase the performance while increasing the computational complexity. For this reason, none of our models use this technique.

We first introduce the relationship between feature maps, default boxes, and ground-truth boxes. As shown in Fig. 4.1, each cell in a feature map is responsible for matching default boxes and ground-truth boxes in a certain area on a resized input image. In our work, that area is named as the *matching area*. For the l^{th} layer of feature map that is of size $h_l \times w_l$, given the input image $h \times w$, the matching area A_l of a single cell in this layer is a $\frac{h}{h_l} \times \frac{w}{w_l}$ region. Since feature maps of our SSD MobileNet-V2 are square of size $s_{fm,l} \times s_{fm,l}$, all matching areas are also square and have a size of $s_l \times s_l$ pixels. For an input resized to $s \times s$, we have $s_l = \frac{s}{s_{fm,l}}$. For example, if the input size of an SSD model is resized to 300×300 pixels and one feature map is of size 3×3, the matching areas of the cells on that feature map will be 100×100. In this subsection, we want to determine the area sizes s_l that best fit our dataset's object size distribution. We will then reduce the required feature map size from these calculated area sizes, given the input image size.

By observing the diagonal distribution of ground-truth boxes of the BDD100K dataset (shown in the bottom left sub-figure of Fig. 3.3,) we know that a large portion of boxes are small. Note that all the objects whose width or height is smaller than the rejection size (22 pixels) are removed, as discussed in Section 3.2.2. We then define small, medium and large vehicles according to their bounding boxes' long edge. A vehicle whose long edge is smaller than 32 pixels is put into *small* category. If the long edge is equal/greater than 32 but smaller than 128 pixels, the vehicle is defined as *medium*. If the long edge is greater than 128 pixels, the vehicle is *large*. Under these criteria, we have 25.16% small vehicles, 63.30% medium vehicle, and 11.52% large vehicles. If we do not apply rejection size and remove very small vehicles, we will have roughly 60% of vehicles smaller than 32 pixels. Therefore, when customizing the feature maps in our SSD models, the capability of detecting small objects affects more to the overall performance than detecting large objects because the small objects appear more frequently. However, for FCW systems, the large objects (*i.e.* close-in vehicles) are more dangerous than small objects

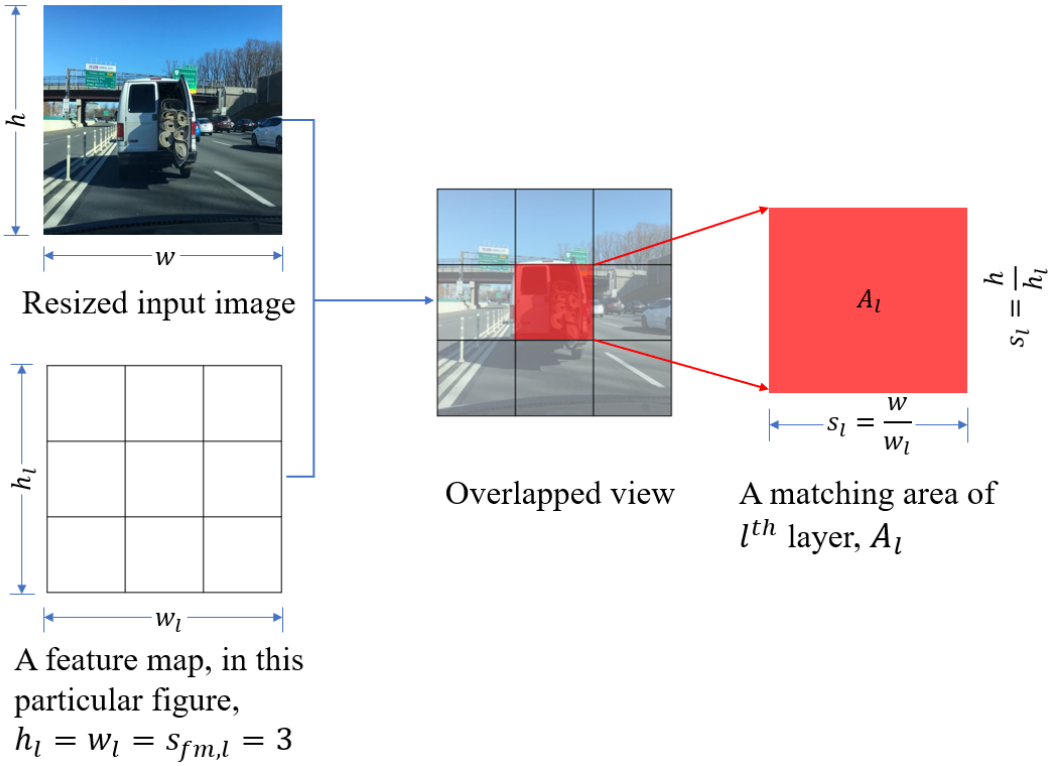

FIGURE 4.1: The matching area defined for SSD

(*i.e.* distant vehicles) and hence need more attention. We group the ground-truth boxes by their diagonal size and design the matching area sizes s_l of feature maps individually, such that the model has as smaller feature maps $s_{fm,l}$ as possible but can still effectively detect each group of boxes. To generate the thresholds of the box grouping method, we propose to use the *Cumulative Distribution Functions (CDF)* of the diagonal length of ground-truth boxes.

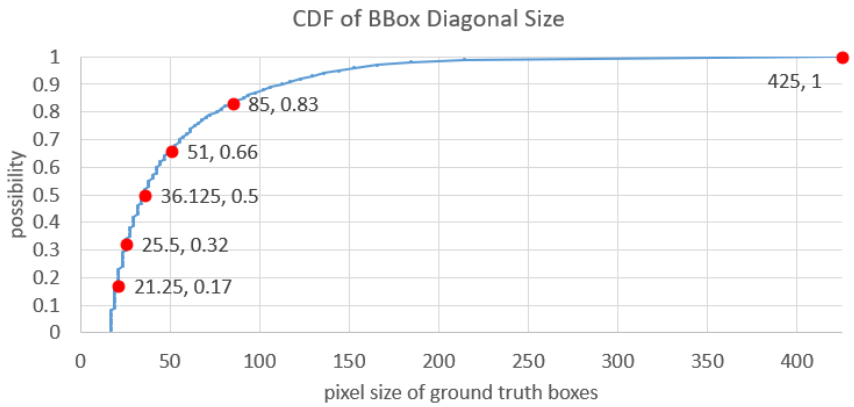

FIGURE 4.2: The CDF of diagonal length of the ground-truth boxes. The red dots denote the selected value for creating matching area sizes of the feature maps.

Fig. 4.2 shows the CDF of the filtered BDD100K dataset, where the input images are resized to 300×300, and ground-truth boxes are resized accordingly. As mentioned above, l denotes the index of the feature map layers in the SSD detection branch. In our SSD model, l ranges from 1 to 6. We divide the closed interval [0.0, 1.0] into six subsets evenly, then take the upper boundaries of each subset as the selected possibility values p_l for the l^{th} layer, which gives $p_l \in \{0.17, 0.32, 0.5, 0.66, 0.83, 1.0\}$. In Fig. 4.2, the red dots are the selected points, while the numbers next to them show the pixel size d_l and possibility p_l. We round the pixel sizes to integers as $d_l \in \{21, 25, 36, 51, 85, 425\}$. Then, we put p_l and d_l in Table 4.1 to calculate the sizes of matching areas. As matching areas A_l are squares of size $s_l \times s_l$, we compute the s_l with $s_l = \frac{d_l}{\sqrt{2}}$. Then we calculate the size of feature maps $s_{fm,l}$ according to the definition of matching area. For example, for an SSD detector with 300×300 input size, we have $s_{fm,l} = 300/s_l$. We round the $s_{fm,l}$ values to the nearest integers for practical design. As a result, the customized multi-scale feature map has six layers, and the shape of these layers are $\{20 \times 20, 17 \times 17, 12 \times 12, 8 \times 8, 5 \times 5, 1 \times 1\}$.

l	1	2	3	4	5	6
p_l	0.17	0.33	0.50	0.67	0.83	1.00
d_l	21	25	36	51	85	425
s_l	15	18	25	36	60	300
$s_{fm,l}$	20	17	12	8	5	1

TABLE 4.1: Feature map sizes generated based on CDF method.

The next step is to implement the SSD feature maps with $s_{fm,l}$ being the size of the feature maps. The customized feature map layers mentioned above cannot use the common 3×3 convolution kernels. To solve this problem, first, we try using larger kernels. For example, when doing the convolution from the first layer (20×20) to the second layer (17×17), if we keep the stride as 1, applying same-padding to the first layer, we need a 5×5 kernel, which needs $17 \times 17 \times (5 \times 5) = 7225$ times of multiplication and addition operations per kernel. For comparison, if we apply 3×3 kernels and set the stride to 2, the shape of the second layer will be 11×11, and the multiplication and addition operations per kernel will be $11 \times 11 \times (3 \times 3) = 1089$. Therefore, our method increases the computation by approximately six times. To alleviate the computation increment, we can use *Spatial Separable Convolution* that replaces the 2D kernel with two 1D kernels, as shown in Fig.4.3. When doing convolution from the first layer to the second layer, instead of using 5×5 kernels, a pair of 5×1 and 1×5 kernels are used. First, it has a convolution from the first feature map layer to an intermediate layer with the 5×1 kernel. It then applies the 1×5 kernel to the convolution from the intermediate layer to the second feature map layer. In this way, the multiplication-addition operations per kernel is reduced to $21 \times 17 \times (5 \times 1) + 17 \times 17 \times (1 \times 5) = 3230$. As a result, after applying Spatial Separable Convolution, the computation is reduced by more than 50%, but it is

still roughly twice larger than regular 3×3 convolution. Note that there are also some other solutions to implement the multi-scale feature maps without increasing computation too much, such as the SSD with *Pooling Pyramid Network (PPN)*[29] and *Deformable Convolution*[8].

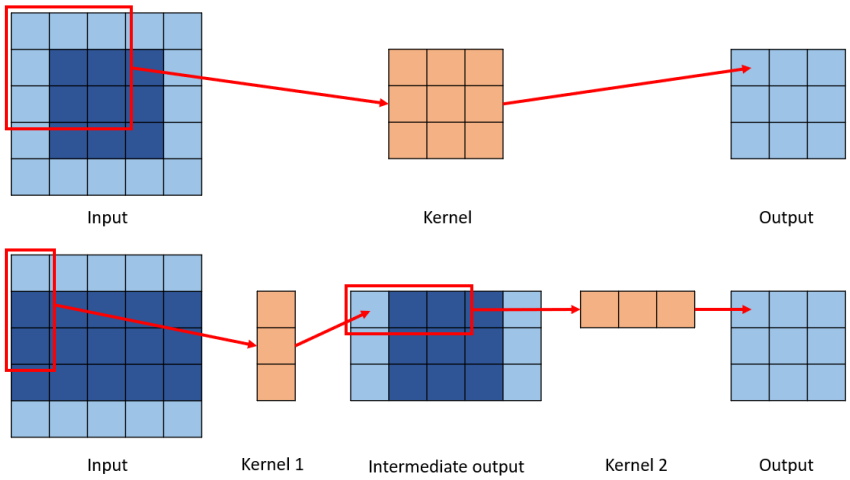

FIGURE 4.3: The comparison between regular and separable 3×3 convolution.
The first row is the regular 3×3 convolution and the second row is the separable 3×3 convolution. The shaded area denotes where the kernel moves through.

We implemented two models with different SSD feature maps. One is a SSD model using Spatial Separable Convolution with feature map sizes of $\{20 \times 20, 17 \times 17, 12 \times 12, 8 \times 8, 5 \times 5, 1 \times 1\}$; this model is named as *SSD-SeparaConv*. The other is using regular convolution with feature map sizes of $\{19 \times 19, 10 \times 10, 5 \times 5, 3 \times 3, 2 \times 2, 1 \times 1\}$, which is named as *SSD-regular*. Both of these models have a 300×300 input size, and both of them use MobileNet-V2 as the backbone network. We train and test these two models on the filtered BDD100K dataset. The performance of SSD-regular and SSD-SeparaConv are shown in Table 4.2. As a result, compared with the SSD-regular model, the SSD-SeparaConv model has higher overall AP and AP on medium/small vehicles. However, it is outperformed by the SSD-regular on the AP of large vehicles. In short, by generating multi-scale feature maps based on the CDF and applying Spatial Separable Convolution, the SSD model can have a better overall performance. In terms of inference time on i7-7800X CPU, the SSD-regular can run at 37.91 fps, while the SSD-SeparaConv only has 28.56 fps. Since large and medium-size vehicles are more important to us, the SSD-SeparaConv model is not suitable for our FCW system, but it is still a successful variant of SSD.

Model Name	AP	AP-L	AP-M	AP-S
SSD-regular	0.17	0.68	0.42	0.04
SSD-SeparaConv	0.20	0.48	0.56	0.19

TABLE 4.2: Performance of SSD implemented with regular and spatial separable convolution.
AP stands for Average Precision, and AP-L is for vehicles larger than 128 pixels, AP-M is for medium vehicles between 32 and 128 pixels, and AP-S is for small vehicles that smaller than 32 pixels.

4.2 Customizing Default Boxes from Gaussian Distribution

This section proposes a strategy to customize the default boxes in SSD based on our dataset analysis results.

In Section 2.4.4, we explained how localization loss is computed with the shape offsets of default boxes, which includes Δcx, Δcy, Δw, and Δh, *i.e.* the offsets along horizontal/vertical axes and width/height offsets. As the shape offsets are predicted and used as a loss in the model, a question arose: Since the networks will learn the shape offsets and make precise predictions eventually, what is the point of customizing the default boxes?

The main answer to this question is that by having an appropriate number and shapes of default boxes, training becomes easier and faster. Let's consider for example, the width and height offsets (Δw, and Δh). Assuming we only have one default box and one ground truth box, their width and height are $[w_1, h_1]$ and $[w_2, h_2]$, respectively, where the w_1 and h_1 are considerably smaller than w_2 and h_2. The localization loss could be large at the beginning of training, and will decrease gradually until $w_1 + \Delta w \approx w_2$ and $h_1 + \Delta h \approx h_2$, *i.e.* the loss reaches one local minimum. This procedure could be very time-consuming, especially when the number of default boxes and ground-truth boxes is large. Hence, customizing default boxes can help reducing the training time and let the localization loss reach a local minimum faster because, at the beginning of the training, the losses Δw, and Δh are already small. A second more intuitive reason is that, through default boxes customization based on prior information from datasets, it is possible to let the global localization loss reaching a better local minimum, thus leading to a model having better performance.

In this section, we discuss how to maximize the efficiency of default boxes when given fixed structure and fixed shape of feature maps. As it will be explained, we propose to obtain the baseline sizes using the Gaussian distribution of the ground-truth boxes, and obtain aspect ratios by applying Gaussian distribution fitting to the aspect ratio histogram. In our experiments, all the models are using MobileNet-V2 as a backbone network, and all the SSD detection branches are using the feature map sizes as in the SSD-regular model found in Section 4.1, that is $\{19 \times 19, 10 \times 10, 5 \times 5,$

$3 \times 3, 2 \times 2, 1 \times 1\}$.

4.2.1 Baseline Size Customization

In our work, the default boxes' baseline sizes are not designed manually, but generated according to IoU between the matching area and the default boxes. The definition and an illustration of IoU are provided in Fig. 2.8. In detection tasks, the IoU is often used for evaluating whether the shape and position of an object are well predicted. Usually, when IoU between the predicted box and the ground truth box is greater than 0.5, we consider it a positive prediction. The higher the IoU, the better the prediction result is.

The key insight is to generate the baseline sizes that give the highest mean IoU between the default boxes and the ground-truth boxes. It can be accomplished in two steps. The first step is grouping the ground-truth boxes by their diagonal size. If the matching areas A_l in the l^{th} layer has a size of $s_l \times s_l$, then all the ground-truth boxes whose width or height are smaller than $\sqrt{2}s_l$ but greater than $\frac{s_l}{\sqrt{2}}$ will be associated with the l^{th} layer, such that, except very elongated boxes (*e.g.* a box with $\frac{w_b}{h_b} = \frac{1}{3}$,) most of ground-truth boxes in l^{th} group have $IoU > 0.5$ with the matching area A_l. When a ground-truth box can be assigned to two or more groups, we assign the box to the group that has the highest layer index l. Fig. 4.4 shows the ground-truth boxes grouping result. We have six groups of ground-truth boxes since we have six multi-scale feature maps. Next, we find a square with the highest mean IoU with the ground-truth boxes for each group. We then take the square's size as the baseline size of the default boxes for the corresponding feature map layer.

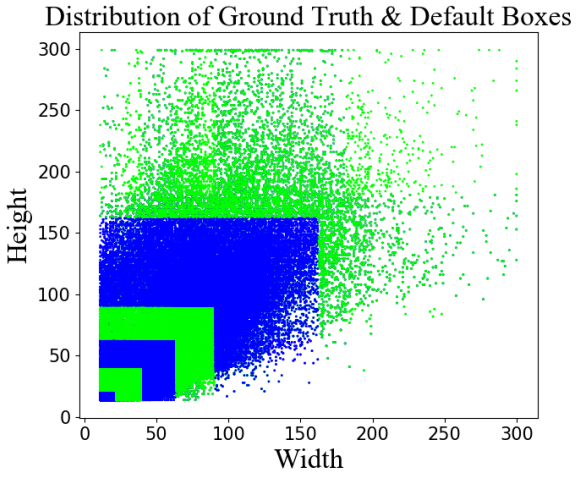

FIGURE 4.4: Ground-truth boxes grouping result of filtered BDD100K dataset.
Each colored stripe denotes a group for the SSD model.

We apply this method to the SSD-regular model (in Table 4.2). With feature map sizes of $\{19 \times 19, 10 \times 10, 5 \times 5, 3 \times 3, 2 \times 2, 1 \times 1\}$, we obtain the baseline sizes for the first to the sixth layer as $s_l \in \{18, 30, 57, 90, 128, 230\}$.

4.2.2 Aspect Ratios Customization

After getting the baseline size of default boxes, the next step is computing the default boxes' aspect ratios according to aspect ratio histogram. Fig. 4.5 shows the aspect ratio histogram of the filtered BDD100K dataset. Since the aspect ratio distribution of the ground truth boxes is likely a Gaussian distribution, we can compute its mean μ and standard deviation σ. As a recap, the original SSD model has the following set of aspect ratios, $a_{ori} \in \{1.0, 0.5, 2.0, 0.333, 3.0\}$. To make a comparison, we also use 5 different aspect ratios, represented as $a_{custom} \in \{\mu - 2\sigma, \mu - \sigma, \mu, \mu + \sigma, \mu + 2\sigma\}$. Each bounding box is roughly responsible for matching ground truth boxes whose aspect ratio is in an interval of $[a_{custom} - \frac{\sigma}{2}, a_{custom} + \frac{\sigma}{2}]$, such that our default boxes can cover the majority of the aspect ratios of the ground truth boxes.

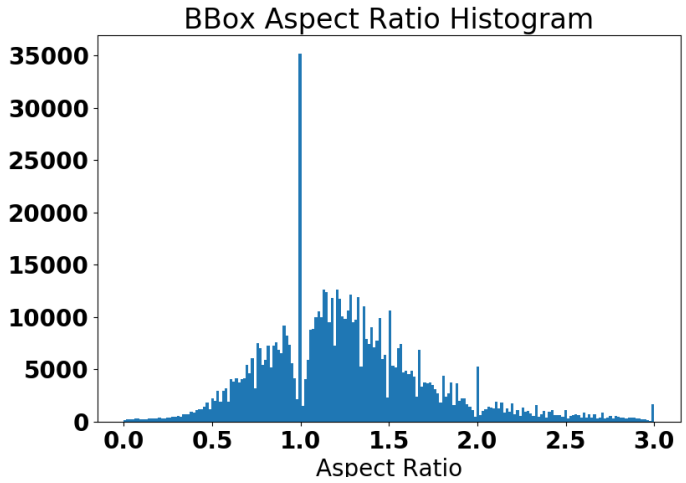

FIGURE 4.5: Aspect ratio histogram of filtered BDD100K dataset.

We compute the mean μ and the standard deviation σ for the training set of the filtered BDD100K dataset, which gives $\mu = 1.31$ and $\sigma = 0.66$. Therefore, the obtained aspect ratios are $a_{custom} \in \{-0.01, 0.65, 1.31, 1.97, 2.63\}$. In Fig. 4.6, we plot the aspect ratios obtained in red vertical lines. This set of aspect ratios is not suitable for default boxes customization because of two problems. First, the aspect ratio -0.01 is negative and cannot be used for default box customization. Second, in a standard Gaussian distribution, in the interval of $[\mu - \sigma, \mu + \sigma]$ should include roughly 68.3% entities. However, we have 84.5% ground truth boxes whose aspect ratio is located in that interval, $[0.65, 1.97]$. If we use the obtained aspect ratios to design default boxes, the default box with $a_{custom} = 1.31$ will be responsible for matching too many ground truth boxes. The default box with $a_{custom} = -0.01$ and $a_{custom} = 2.63$

could be useless as they can only match very thin or very wide ground truth boxes, which represent less than 2% of the ground-truth boxes in the filtered BDD100K dataset. The pulses and valleys (aspect ratio bias of datasets mentioned in Section 3.2.3) cause these two problems, as they increase the standard deviation significantly.

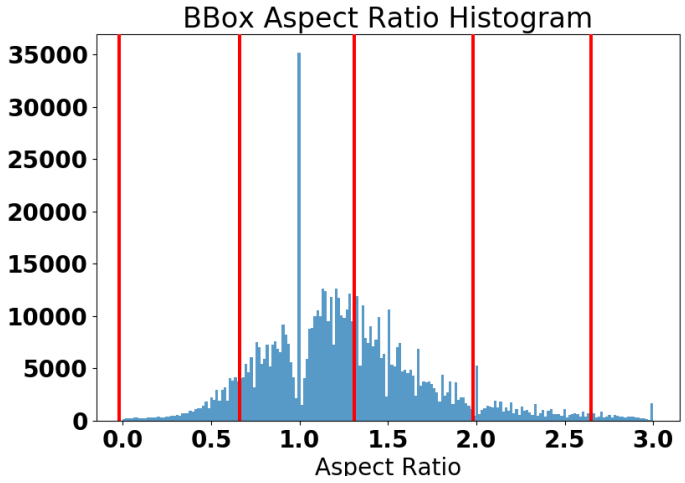

FIGURE 4.6: Aspect ratios obtained for filtered BDD100K dataset.
The five red vertical lines show the location of the selected value in the histogram.

To alleviate the impact of the aspect ratio bias, we propose a simple approach, which is processing the histograms with data smoothing. The key idea is to take the aspect ratio histogram as a signal, apply Low-pass filtering to smooth the histogram, then use Gaussian Probability Distribution Fitting (Gaussian PDF) to estimate the probability distribution of the aspect ratio. More details are given in Appendix C.

As a result, the customized aspect ratios are $a_{custom} \in \{0.32, 0.8, 1.28, 1.76, 2.24\}$. To compare with aspect ratio before data smoothing (Fig.4.6), we plot the selected aspect ratios after data smoothing and Gaussian PDF in Fig. 4.7.

Taking the feature maps sizes and customized baseline sizes obtained in the previous section, and the aspect ratios generated from the Gaussian PDF figure, we designed new default boxes, named this model as *SSD-GPDF* for the experiments later. We show the default box setups of the SSD-GPDF model in Table 4.3.

Baseline Sizes	18, 30, 57, 90, 128, 230
Aspect Ratios	0.32, 0.8, 1.28, 1.76, 2.24

TABLE 4.3: Default boxes setups for the SSD-GPDF model.

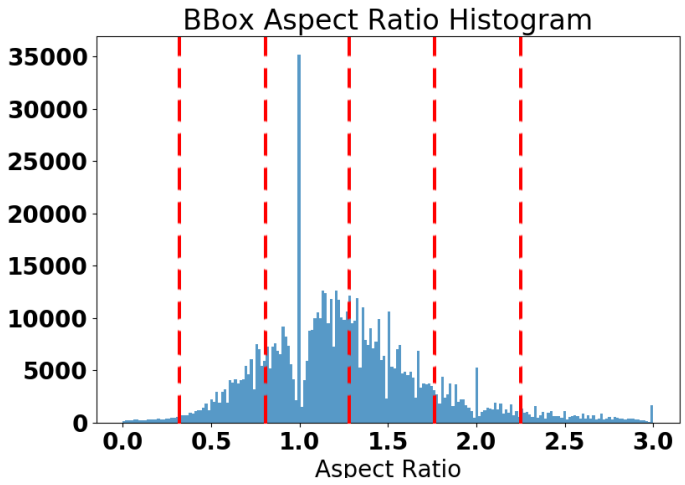

FIGURE 4.7: Aspect ratios obtained after histogram smoothing and Gaussian PDF.
The dashed red line are the newly obtained aspect ratios after histogram smoothing and Gaussian PDF.

4.3 Obtain Default Boxes from Layer-wise K-Means Clustering

In the original SSD and our SSD-GPDF model (from the previous section), the default boxes are customized into two steps: the first step is computing baseline sizes, and the second step is obtaining aspect ratios. In principle, a better strategy should be to determine the baseline sizes and the aspect ratios in one shot. In this section, we achieve this objective by using K-Means clustering.

In the two models mentioned above, the number of default boxes per feature map cell is the same. As the feature maps are responsible for predicting objects for which the sizes are similar to the matching area, the SSD structure does not have a preference for object size. Generally, having no preference is beneficial for general object detection. However, it leads to a waste of calculation if some objects within a specific range of sizes are not present. In our application, the vehicles in our dataset exhibit a specific distribution of aspect ratios which should be taken into account.

In YOLO-V2 [48], a K-Means clustering method is used for finding both the shape and the number of the *Anchor Boxes*; These are equivalent to the default boxes in SSD. Instead of using original K-Means clustering, whose score is a sum of squared distances from samples to centroids, YOLO-V2 uses a distance score based on IoU for clustering such that the trained detector produces predictions with high IoU. The distance score is shown in Equation 4.1.

$$d(box, centroid) = 1 - IoU(box, centroid) \tag{4.1}$$

Inspired by YOLO-V2, our first try uses the regular K-Means clustering with the IoU-based distance score to generate both the shape and number of default boxes for our SSD model. We apply the regular K-Means clustering on the filtered BDD100K dataset's training set and take the *centroids* (*i.e.* centers of each cluster) as the default boxes of our SSD detector. The computed default boxes are shown in Fig. 4.8. It turns out that the default boxes do not cover the large ground-truth boxes because of the sparsity of data for those objects, especially in the top right part of the figure. Since the close-in vehicles are essential for FCW systems, it requires our SSD detector to have more default boxes around large objects. Therefore, the regular K-Means does not fit this requirement. None the less, we built a model with regular K-Means clustering, named as *SSD-regular-KM*, which is used for comparison in our experiments.

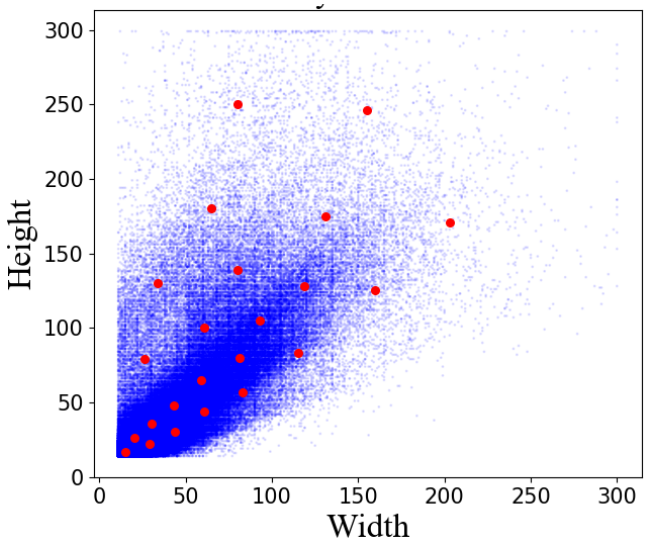

FIGURE 4.8: Computed default boxes from regular K-Means clustering.
The blue scatter denotes ground-truth boxes in the training set of the filtered BDD100K dataset. The red dots are the default boxes computed with regular K-Means clustering.

To obtain the optimal shapes and number of default boxes for each feature map layer in SSD, we propose to apply *Layer-wise K-Means* to SSD detectors. The Layer-wise K-Means clustering is composed of two steps, layer-wise boxes grouping and K-Means clustering.

4.3.1 Layer-wise ground-truth boxes Grouping

Our method's first step is layer-wise box grouping, which divides the whole training set of the filtered BDD100K dataset into six groups since we have six multi-scale feature maps in SSD.

We propose an IoU-based constraint to group ground-truth boxes. In this method, all the ground-truth boxes are grouped to the layer with whose matching area it produces the highest IoU. Fig.4.9 shows the grouping result of this method.

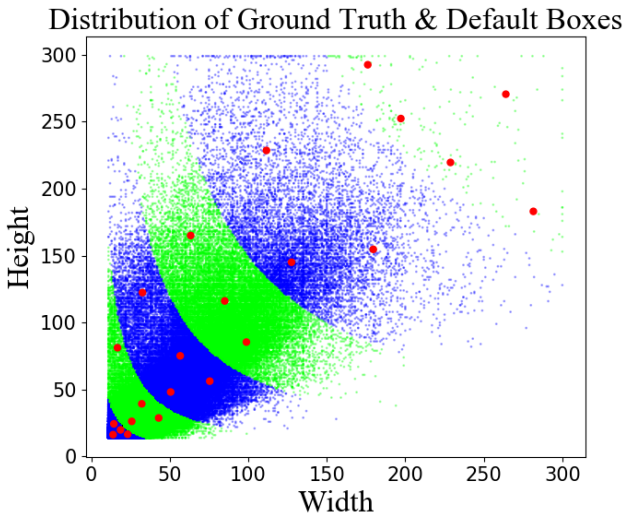

FIGURE 4.9: IoU-based ground-truth boxes grouping method.
The blue and green scatters denote the box groups from the first to the last layer of SSD feature maps. The red dots are the K-Means clustering result in one of our experiments.

4.3.2 Layer-wise K-Means Clustering

We apply IoU-based K-Means Clustering to boxes group by group to get the centroids for each group of boxes after layer-wise box grouping. The obtained centroids will be used as the default boxes for the SSD detection branch.

First, we apply an identical amount of K-Means centroids to all the six feature map layers. We let $K = 5$ when running clustering for each group of boxes, such that it produces the same number of default boxes as the original SSD model. The result is shown in Fig. 4.10. Compared with the result of regular K-Means in Fig. 4.8, this time we have more centroids close to large objects. According to our experiments, if we increase the value of K and produce more default boxes, the performance of SSD detector will be improved, and the computation is also increased.

We noticed that for each layer, the optimal number of centroids might differ. Therefore, we propose a default box optimality cost to get the optimal numbers and shapes of default boxes. The optimality cost E is defined as follows:

$$E = \frac{s}{n} \times \frac{p}{c} = \frac{\sum_{l=1}^{6} s_{km,l}}{n} \times \frac{\sum_{l=1}^{6} c_l \times s_{fm,l}^2}{\sum_{l=1}^{6} c_l} \tag{4.2}$$

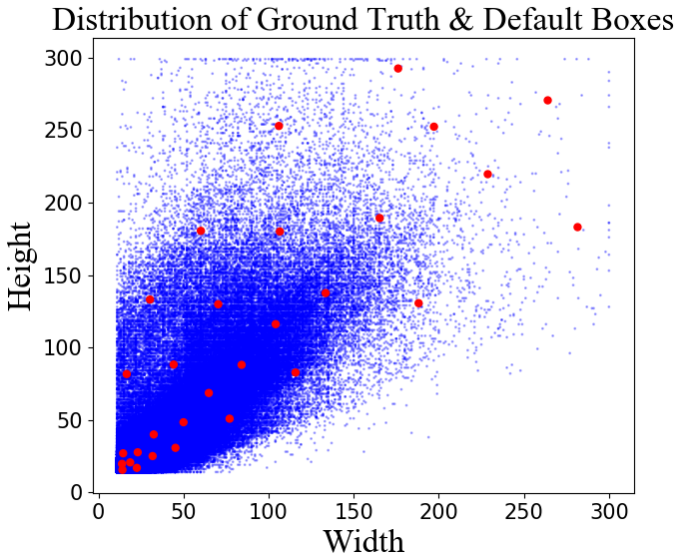

FIGURE 4.10: Layer-wise K-Means with fixed number of centroids.
The blue scatters are ground-truth boxes in the filtered BDD100K dataset, and the red dots denote clustering centroids (*i.e.* default boxes). For each layer, we have 5 centroids.

where s is the total K-Means clustering score, n is the number of ground-truth boxes in the training set, c_l is the number of centroids for the l^{th} layer, $s_{fm,l} \times s_{fm,l}$ is the shape of the l^{th} layer of SSD feature map. p denotes the total number of default boxes in all the six SSD feature maps. $s_{km,l}$ is the score of IoU-based K-Means that corresponds to the l^{th} layer, defined by:

$$s_{km,l} = \sum_{i=1}^{c_l} \sum_{b \in N_l} d(b, \mu_{i,l}) = \sum_{i=1}^{c_l} \sum_{b \in N_l} [1 - IoU(b, \mu_{i,l})] \tag{4.3}$$

where $b \in N_l$ denotes all the ground-truth boxes in the l^{th} layer, $\mu_{i,j}$ is the i^{th} centroid in the l^{th} layer.

In equation E, $\frac{s}{n}$ stands for the average distance from ground-truth boxes to the centroids. The term $\frac{s}{n \times c}$ in the equation can be seen as the average K-Means score per centroid per default box, the smaller it is, the better the centroids fit the data. p roughly indicates the computation requirement as the number of raw predictions in SSD models equal to the total number of default boxes. Consequently, the smaller the E is, the better default boxes we have. Note that reducing the $\frac{s}{n \times c}$ by increasing c, *i.e.* using more centroids, causes a larger p, produces more predictions, and brings us higher computation.

Since we have six groups of boxes, for each SSD model there are six K-Means clustering. In each clustering, we increase the K value gradually from 2 to 10 and apply the Elbow method [62] to find the optimal K value by finding the a turning point on the curve of K-Means score $s_{km,l}$. More details about the Elbow method can

be found in Appendix D. Then, we compute the efficiency cost of our models, which is shown in Fig. 4.11. We choose 0.75 as the Elbow method threshold for the IoU-based grouping method, which give relatively lower efficiency cost and preserve adequate number of default boxes for detection.

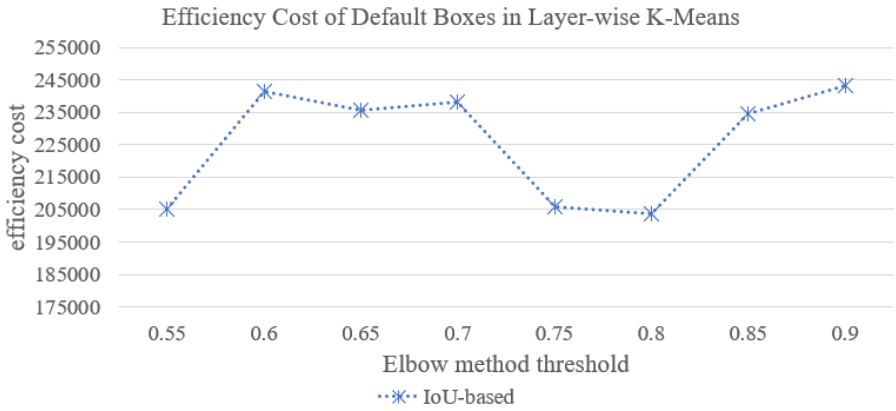

FIGURE 4.11: Efficiency cost of Layer-wise K-Means Clustering.

In Fig. 4.12, we compare the clustering results between regular K-Means and our method. We let the regular K-Means clustering having the same number of centroids as in our method. The default boxes in our method covers more large objects. We name our SSD model that uses IoU-based boxes grouping and Layer-wise K-Means clustering as *SSD-LW-KM* for later experiments. Note that our method is designed for getting more effective but fewer default boxes. More default boxes could be used for getting better performance if the computation is not a major concern.

4.3.3 Default Boxes Comparison

In this section, we compare the obtained default boxes in three steps. First, we present the default boxes obtained using different methods in a table. Second, we visualize the IoU results between the default boxes and all possible boxes by drawing a complete heat maps. Third, we draw IoU heat maps with practical data (filtered BDD100K dataset).

Default Boxes Table

We list the default box sizes of our four different models in Table 4.4. All the these models are generated from the filtered BDD100K dataset (that includes the objects greater than 22 pixels).

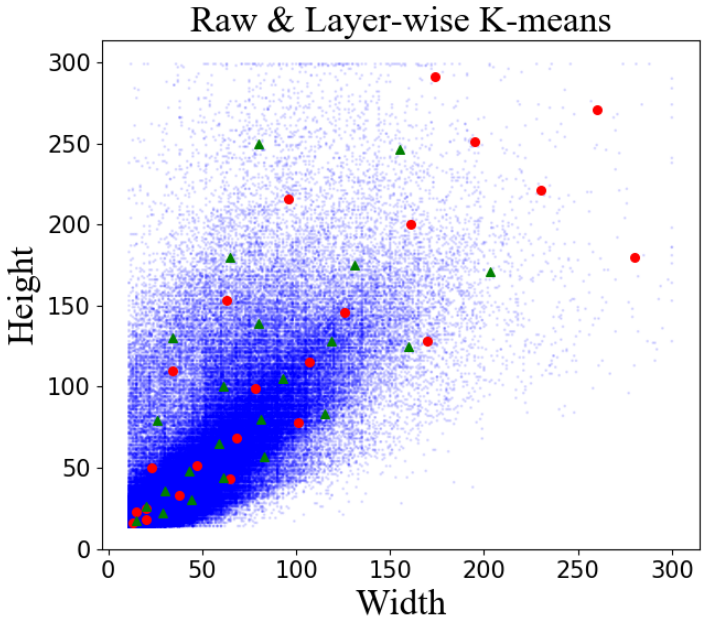

FIGURE 4.12: Clustering result of regular K-Means and our method that uses IoU-based constraint.
The green triangles denote the regular K-Means clustering result and the red dots are the result of our method.

Complete IoU Heat Map

To assess the different default boxes' impact, we created heat maps showing the IoU of all possible bounding boxes with its closest default box. We consider input boxes from 1×1 to 300×300 pixels as the input size of our SSD models is 300×300. For each box B_i, we compute its IoU scores over all the default boxes P_j in the model, then keep the highest IoU score as the final output H_i, as shown by the following equation.

$$H_i = H(B_i) = \max(IoU(B_i, P_{j=1}), ..., IoU(B_i, P_{j=c})), i \in [1, n] \tag{4.4}$$

In the equation, n is the total number of input boxes, c is the number of centroids of Layer-wise K-Means clustering (*i.e.* the generated default boxes). Fig. 4.13 shows the IoU heat maps of our four models. The horizontal and vertical axes are object height and width, in pixels. For each pair of default box and input box, the higher the IoU is, the brighter the color is.

Table 4.5 presents the mean IoU (mIoU) of these models. We notice that, when using all possible input boxes, the original SSD model has the highest mean IoU. This suggests the original SSD is better for general object detection task.

In the IoU heat map of the SSD-original model, for objects smaller than 50×50 pixels, the IoU scores are nearly zero. Therefore, the network has to make predictions

Model	Layer	Default Boxes [width, height]
SSD-original	1	[51, 51],[72, 36],[36, 72],[88, 29],[29, 88],[67, 67]*
	2	[89, 89],[126, 63],[63, 126],[155, 51],[51, 155],[106, 106]*
	3	[128, 128],[180, 90],[90, 180],[145, 145]*,[221, 73],[73, 221]
	4	[166, 166],[235, 117],[117, 235],[184, 184]*,[288, 96],[96, 288]
	5	[204, 204],[289, 144],[144, 289],[222, 222]*,[354, 118],[118, 354]
	6	[243, 243],[343, 171],[171, 343],[270, 270]*,[421, 140],[140, 421]
SSD-GPDF	1	[15, 12],[12, 15],[10, 18],[9, 20],[24, 8]
	2	[29, 23],[23, 29],[19, 34],[17, 38],[45, 14]
	3	[57, 46],[45, 58],[38, 68],[34, 76],[90, 29]
	4	[95, 76],[75, 96],[64, 113],[57, 127],[150, 48]
	5	[143, 114],[113, 144],[96, 169],[85, 191],[225, 72]
	6	[285, 228],[225, 288],[192, 338],[170, 382],[450, 144]
SSD-regular-KM	1	[17,15]
	2	[26,20],[36,30],[22,29],[30,44]
	3	[65,59],[44,61],[57,83],[48,43],[79,26]
	4	[80,81],[130,34],[100,61],[105,93],[139,80],[83,115]
	5	[180,65],[250,80],[125,160],[171,203],[175,131],[128,119]
	6	[246,155]
SSD-LW-KM	1	[16,13],[23,15],[18,20]
	2	[25,20],[50,23],[33,38]
	3	[51,47],[68,68],[43,65],[110,34]
	4	[115,107],[153,63],[99,78],[78,101]
	5	[146,126],[200,161],[216,96],[128,170]
	6	[221,230],[180,280],[291,174],[271,260],[251,195]

TABLE 4.4: Default boxes of different SSD models.
* denotes additional default boxes in the original SSD model that has aspect ratio $a = 1.0$.

among small objects with the closest (smallest) default boxes that are considerably larger than the object size. During training, the IoU scores between small objects and the predictions from the smallest default boxes remain low during training. Hence, the network produces large offset loss for small objects. We believe that not having small default boxes could be one of the reasons why it is challenging to train the original SSD model with small objects, even if having a long training time and a large dataset. On the other hand, the middle and the top right part of the figure are bright, which explains why the original SSD model get trained fast and performs well on medium and large objects.

The SSD-GPDF model has the same number of aspect ratios of default boxes as in the original SSD. The SSD-GPDF model has slightly higher mIoU-S than the original SSD since its default boxes cover small objects better. However, there is a vast gap between medium and large objects on the heat map, which represents the model could be ineffective when predicting large objects. Given Fig. 4.12 and the SSD-regular-KM figure, we know the SSD-regular-KM fit the training data well except for large objects. Consequently, compared with the SSD-LW-KM, its mIoU-L is

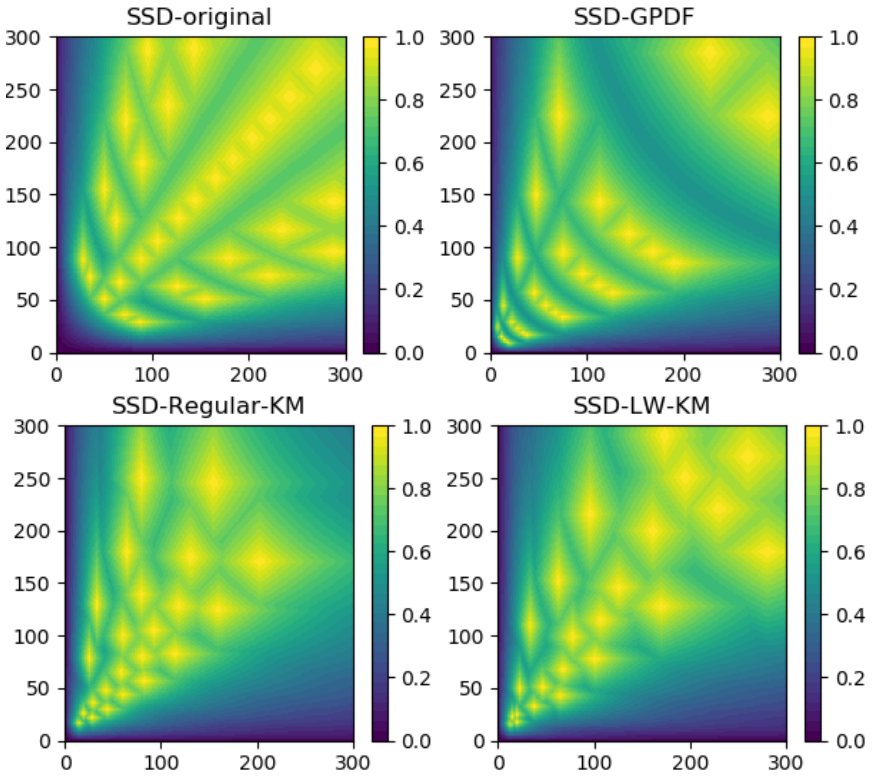

FIGURE 4.13: Complete IoU heat maps of different models.

Model	mIoU-S	mIoU-M	mIoU-L	mIoU
SSD-original	0.3196	**0.7818**	**0.8451**	**0.7093**
SSD-GPDF	**0.3618**	0.7049	0.7075	0.6364
SSD-regular-KM	0.3288	0.6819	0.7317	0.6270
SSD-LW-KM	0.3121	0.6588	0.8425	0.6492

TABLE 4.5: Mean IoU in the complete IoU heat maps.
mIoU is mean IoU over all possible bounding boxes; mIoU-S, mIoU-M, and mIoU-L are mIoU over small, medium and large ground-truth boxes. The largest value for each mIoU category is bold.

lower.

IoU Heat Map of Practical Data

We performed the IoU heat map experiment on the filtered BDD100K dataset for our four models. These heat maps are shown in Fig. 4.14.

Visually, we have more bright yellow scatters on the figures of SSD-regular-KM and our SSD-LW-KM models. To prove it statistically, we compute the mIoU-L, mIoU-M, mIoU-S and overall mIoU for these models, and Table 4.6 shows the result. It turns out that our SSD-LW-KM model provide considerably higher mIoU compared with the original SSD model.

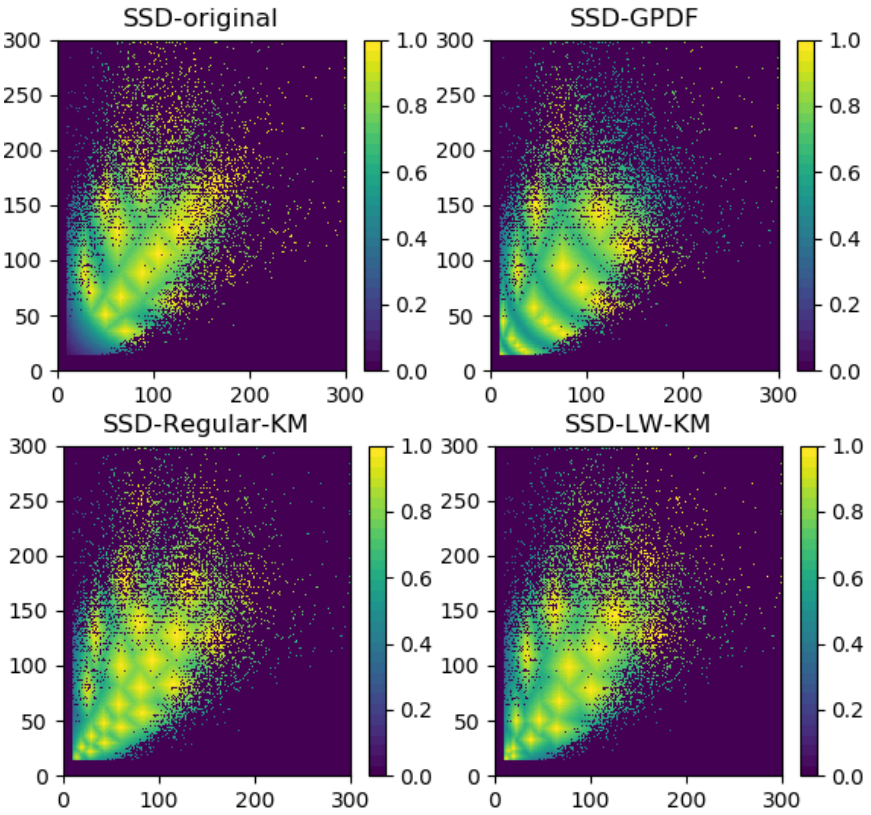

FIGURE 4.14: IoU heat maps on the filtered BDD100K dataset.

Model	mIoU-S	mIoU-M	mIoU-L	mIoU
SSD-original	0.2154	0.7546	**0.8661**	0.4230
SSD-GPDF	0.7197	0.7396	0.6796	0.7265
SSD-regular-KM	0.7810	**0.8263**	0.8338	0.7984
SSD-LW-KM	**0.8094**	0.7988	0.8302	**0.8017**

TABLE 4.6: Mean IoU in the heat maps of the filtered BDD100K dataset.

Chapter 5

Experimental Results of Vehicle Detection

In this chapter, we present our experiments on vehicle detection, including the dataset we used, a list of the SSD models defined in Chapter 4, the training and evaluation setups, and the experimental results.

5.1 Dataset

We used a large and publicly available dataset, the BDD100K [65] dataset for our vehicle detection experiments. As mentioned in Chapter 3, we kept only the vehicles (*i.e.* classes of car, truck, and bus) and we did some pre-processing to this dataset and named this pre-processed dataset as *filtered BDD100K* in which all the vehicles are larger than 22 pixels.

5.2 Implementation Details

5.2.1 List of Models

In the following experiments, we compare four major models: The first model is *SSD-original*, the original SSD model; The second one is *SSD-GPDF* that uses our Gaussian PDF method for default boxes customization (Section 4.2). The third model is *SSD-LW-KM* that uses our Layer-wise K-Means method (Section 4.3). Additionally, we include the *SSD-regular-KM* model in our experiments, whose default boxes are generated by regular K-Means clustering (introduced at the beginning of Section 4.3). All of these four models are using MobileNet-V2 as their backbone network. The shapes of their SSD feature maps are $\{19 \times 19, 10 \times 10, 5 \times 5, 3 \times 3, 2 \times 2, 1 \times 1\}$.

The number of default boxes per SSD feature map layer and the total number of default boxes per model are shown in Table 5.1. As the SSD-GPDF model changes the aspect ratios of default boxes only, it has the same number of default boxes as the SSD-original model, that is 2500. All the other models have fewer default boxes

compared with the original SSD-original model.

Model	L_{15}	L_{19}	L_{20}	L_{21}	L_{22}	L_{23}	p
SSD-original	5	5	5	5	5	5	2500
SSD-GPDF	5	5	5	5	5	5	2500
SSD-regular-KM	1	4	5	6	6	1	965
SSD-LW-KM	3	3	4	4	4	5	1540

TABLE 5.1: The number of default boxes per layer in different models. L_N denotes the N^{th} convolutional layer in our network. p is the total number of default boxes in a model (defined in Equation 4.2).

5.2.2 Training Configurations

We do not use the popular Adam optimizer [34] for gradient descent in our models' training, since Adam optimizer occasionally fails to converge to an optimal solution and sometimes provides performance that is worse than the Stochastic Gradient Descent (SGD) [33, 43]. We trained our models with RMSprop [24], a momentum gradient descent method proposed by Geoffrey Hinton. Generally, with RMSprop, the training time is longer than with Adam, but the performance is better and more stable. Since we are using the RMSprop optimizer, the batch size cannot be too small. We set the batch size to 24, the learning rate to 0.006. The models are trained with $840K$ iterations, during which the learning rate is multiplied by 0.95 every $70K$ iterations. As applied in the original SSD, we use the same multi-task loss, hard-negative mining, as well as the same default and ground truth boxes matching strategy for the rest of models.

5.3 Evaluation Results

5.3.1 Evaluation Configurations

As a recap, in Chapter 4, we divided vehicles into three categories according to the length of their longest edge. A vehicle whose long edge is smaller than 32 pixels is put into *small* category. If the long edge is equal/greater than 32 but smaller than 128 pixels, the vehicle is defined as *medium*. If the long edge is greater than 128 pixels, the vehicle is *large*.

We evaluate our four models with following criteria: Precision-Recall curve, Average Precision and Frame Rate.

5.3.2 Precision and Recall

Precision and *recall* are frequently used metrics for identifying whether an object detector is good or not. The precision and recall are computed from the *confusion matrix* of our object detection task (shown in Fig. 2.7, and Equation 2.2 and 2.3). In short, if the precision is high, the predicted "vehicles" are more likely to be actual vehicles. If the recall is high, few vehicles are missed when making predictions. An ideal object detector should be able to reach the value of 1.0 for both precision and recall. However, it is impracticable to achieve this goal because of many factors. A detector with 1.0 precision but low recall could miss many vehicles, and a detector with 1.0 recall but low precision make too many predictions that are not vehicles. Both of these two cases are not what we need.

In our work, a Precision-Recall curve is used to compare our SSD detectors' performance, where the horizontal axis is recall and the vertical axis is precision. The curve can help us choose the confidence thresholds for each model to get a decent detector that has high precision and recall at the same time.

The Precision-Recall figures of the original and our SSD models are shown in Fig. 5.1, which includes the performance of detecting large, medium, and small objects, and the overall performance that combines them. We provide two original SSD models with different input size of 224×224 and 300×300, which are presented as *ssd-original-224* and *ssd-original-300*. The input size of other three models are 300×300. The blue and red stars in the figures show the points on the SSD-original-300 and SSD-LW-KM curves that give the best balance between precision and recall.

The performance of all the SSD models on detecting medium and large objects are generally acceptable. In the Precision-Recall (large) figure, other models does not have a noticeable difference except for the SSD-regular-KM model. The result suggests that it is not challenging to build SSD models to detect large objects. Interestingly, even the original ssd-original-224 model outperforms the SSD-regular-KM model when detecting large objects. It shows that the default boxes in SSD-regular-KM are less effective than the original default boxes. Hence, default boxes customization is essential for building SSD detectors. Our SSD-LW-KM model outperforms all the other models on overall performance by a considerable margin.

In the Precision-Recall figure of small objects, all of these SSD models present poor performances, which suggest that training SSD detectors with small objects is a challenging task. Nevertheless, our SSD-GPDF and SSD-LW-KM models perform better than the SSD-original-300 and the SSD-original-224 models.

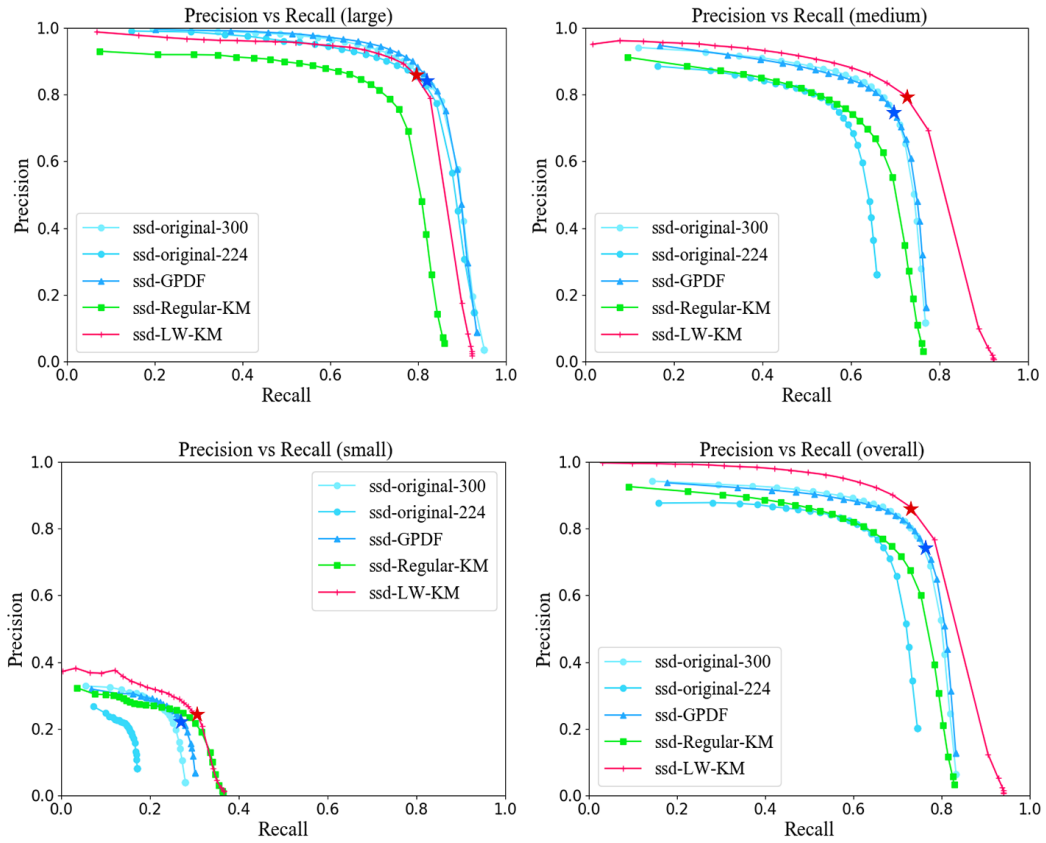

FIGURE 5.1: Precision-Recall curve of different SSD models.

The last figure in Fig. 5.1 shows the overall performance of detecting all the vehicles. As a result, the performance our SSD-LW-KM model has higher precision and recall compared with all the other SSD models.

Interestingly, the SSD-regular-KM model has good centroids in K-Means clustering, but the default boxes built with those centroids did not help improve the performance. The main reason is that we assigned the default boxes in the SSD-regular-KM model in the same way as in our SSD-LW-KM model such that it only has one default box for the first feature map and one for the last layer (shown in the Table 5.1.) Therefore, during training, the medium-size default boxes are matched with small, medium, and large ground truth boxes and produce large localization loss, making the whole network hard to get trained.

We choose confidence thresholds for the models from the Precision-Recall curves. Because of the medium-range vehicles are the most important to our FCW system, we value the Precision-Recall curve of medium vehicles most. We chose to use our SSD-LW-KM model with confidence threshold 0.15. The corresponding precision and recall are shown in Table 5.2.

Model Name	Medium (%)		Large (%)		Overall (%)		thresh.
	prec.	rec.	prec.	rec.	prec.	rec.	
SSD-LW-KM	79.9	72.1	86.2	79.1	86.5	72.9	0.15

TABLE 5.2: The precision, recall and chosen confidence thresholds of our model.

5.3.3 Average Precision

Average Precision (AP) (introduced in in Section 2.3.2) is another frequently used standard to evaluate object detectors. Since we only have a single mixed class, vehicle, we compute the AP of the class "vehicle" but not the mean AP of all the sub-classes (car, truck, bus, etc).

We compute the overall *AP* and the AP for large, medium, and small vehicles respectively, as *AP-L*, *AP-M*, and *AP-S*. The result is shown in Table 5.3, in each column, the largest value is highlighted. Our SSD-GPDF model has the highest AP on detecting large objects but does not outperform the two SSD-original models with a great margin. Our SSD-LW-KM model performs best on detecting medium objects, and it has the highest overall AP.

Model Name	AP-L	AP-M	AP-S	AP
SSD-original-224	0.8339	0.5364	0.0408	0.6122
SSD-original-300	0.8529	0.6595	0.0802	0.7189
SSD-GPDF	**0.8576**	0.6615	0.0851	0.7203
SSD-regular-KM	0.7153	0.5902	**0.0913**	0.6647
SSD-LW-KM	0.7978	**0.7118**	0.0702	**0.7636**

TABLE 5.3: Average precision over different sizes of vehicles.

5.3.4 Frame Rate

We check the *Frame Rate* of our models in *frame per second (FPS)*. The experiment includes two original SSD models, SSD-original-224 and SSD-original-300, and our best model, SSD-LW-KM. The experiments were executed on two processors. The first processor is a single GTX1080Ti GPU with a fixed 9 GB memory. The second one is an i7-7800X CPU that has a base frequency of 3.5GHz. We compute the inference time of our SSD models for both with and without the Non-Maximum Suppression (NMS) (a post-processing algorithm to remove overlapped raw predictions). In our case, the NMS is always processed by CPU.

Fig. 5.2 shows the frame rate of our models running on CPU and GPU. With NMS processing time considered, our SSD-LW-KM model has a higher average frame rate on GPU compared with both the SSD-original-300 and SSD-original-224 models. If

we exclude NMS processing time, our SSD-LW-KM model and the SSD-original-300 model has a similar frame rate. The findings suggest our method successfully reduces the number of predictions and slightly increases the frame rate on GPU.

Looking at the frame rate on CPU, whether with or without NMS, our SSD-LW-KM model outperforms the SSD-original-300 model, and its frame rate is more stable than the two SSD-original models. Since our goal is to build an FCW system for mobile devices that only have CPU, CPU's result is more important to us. Therefore, we conclude that our SSD-LW-KM model is better than the original SSD model in terms of precision/recall, AP, and frame rate.

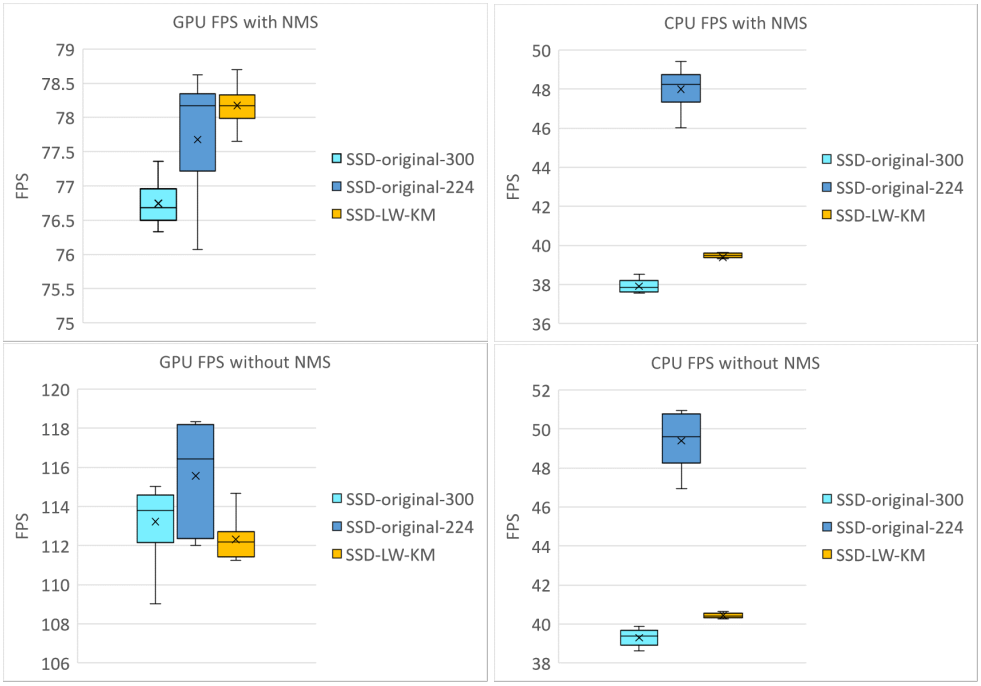

FIGURE 5.2: Frame Rate of models when running on single GPU and CPU, with and without NMS.

We present the minimum, maximum and average time of processing each image frame, for both running on GPU and CPU, in Table 5.4 and Table 5.5. When running the models on GPU, the time of post-processing (mainly NMS) is nearly half of the prediction. Therefore, Post-processing is a bottleneck of SSD detectors.

	GPU Prediction (ms)			CPU Post-processing (ms)
Name	Min	Max	Avg	Avg
SSD-original-300	8.69	9.17	8.83	4.19
SSD-LW-KM	8.72	8.99	8.90	3.07

TABLE 5.4: Processing time per image frame on GPU.

	CPU Prediction (ms)			CPU Post-processing (ms)
Name	Min	Max	Avg	Avg
SSD-original-300	25.07	25.89	25.45	0.93
SSD-LW-KM	24.40	25.07	24.73	0.65

TABLE 5.5: Processing time per image frame on CPU.

5.3.5 Failure Cases

This section presents some failure cases of vehicle detection of our SSD-LW-KM model on both the filtered BDD100K dataset and the Viewnyx dataset.

Fig. 5.3 shows some failure cases of our SSD-LW-KM model on the filtered BDD100K and the Viewnyx dataset, where the green bounding boxes show the detected vehicles. In the first three pictures of Fig. 5.3, the failure cases include uncommon vehicles (construction machines), and vehicles on night time and highly blurred images. For these three failure cases, the problem could be, to some extent, alleviated by providing sufficient data for those situations. What caught our attention is the fourth picture, in which a semi-transparent cloth pattern covers the car in the bottom-right corner. That car is detected poorly, but could be easily recognized by human observers. Such a problem is also reported in other researches. For example, [46] shows that deep learning classifiers can be easily fooled by some delicately designed noise patterns.

FIGURE 5.3: Failure cases of our SSD-LW-KM model.

Chapter 6

Forward-leading Vehicle Detection Acceleration

In this chapter, we detect *forward-leading vehicles* with video sequence as input. In Chapter 5 we concluded that our SSD-LW-KM model is the most successful object detector among all of our attempts to improve SSD MobileNet-V2 object detectors. Our object detector predicts all the possible vehicles that appeared in each frame, but here we want to identify the forward-leading vehicle specifically. As mentioned in Section 3.3, our detector will be implemented on a regular smartphone to perform forward-leading vehicle detection. Though our SSD-LW-KM model is efficient, its computational cost is still high for the target device. It requires us to further reduce the computation of our solution.

In recent years, many video object detectors have been proposed for mobile devices. In *Fast YOLO* [57], a motion driven probability map is applied to update key-frame and make predictions. The Fast YOLO detector reaches 17.85 FPS on an NVIDIA Jetson TX1 embedded system. Mason *et al.* [40] built a video object detector with two feature extractors with different running speeds and recognition capacities, together with a convolutional LSTM layer that is to store the visual memory. However, the small feature extractor and LSTM layer in Mason's work still set a high computation and memory requirement for regular smartphones. We propose to apply object tracking to accelerate the detection, such that the whole application has a high frame rate on our mobile device.

From Section 6.1 to 6.3 we briefly introduce our basic assumptions and definition about forward-leading vehicles, as well as a fine-tuning technique to adapt our SSD-LW-KM model to our Viewnyx dataset. In Section 6.4, we combine an object detector with an object tracker, where the detection branch provides a precise prediction of the leading vehicle, and the tracking branch provides a rough prediction.

6.1 Basic Assumptions

We first introduce our basic assumptions about forward-leading vehicles.

The first assumption is that the orientation of the forward-leading vehicle is nearly identical to the orientation of the vehicle on which the camera is mounted since these two vehicles are moving forward along the same lane most of the time. In this case, the leading vehicle's horizontal shifting speed in the camera frames is considerably small, except the two vehicles are making a sharp turning, or the distance between them is large. It leads to our second assumption that, on the trajectory of the leading vehicle in consecutive frames, the bounding boxes have very few sudden changes. To validate this assumption, we choose to use the IoU of vehicles' bounding boxes in two successive frames as our evaluation metric. The higher the IoU is, the fewer positional changes the bounding boxes have.

We extract a long video sequence from our Viewnyx dataset and compute the IoU between leading vehicles in every two consecutive frames to validate our assumption. Fig. 6.1 shows the result, and the mean IoU is 0.5046. By checking the frames manually, we know that the IoU is very high in some frames because the leading vehicle is close to the camera. The low IoU values are caused by vehicles far away, especially when the vehicles are turning. Several zeros and a few sudden changes in the curve occur when the labeled leading vehicles in the two consecutive frames are not the same vehicle. Therefore, under most circumstances, our assumptions are reasonable.

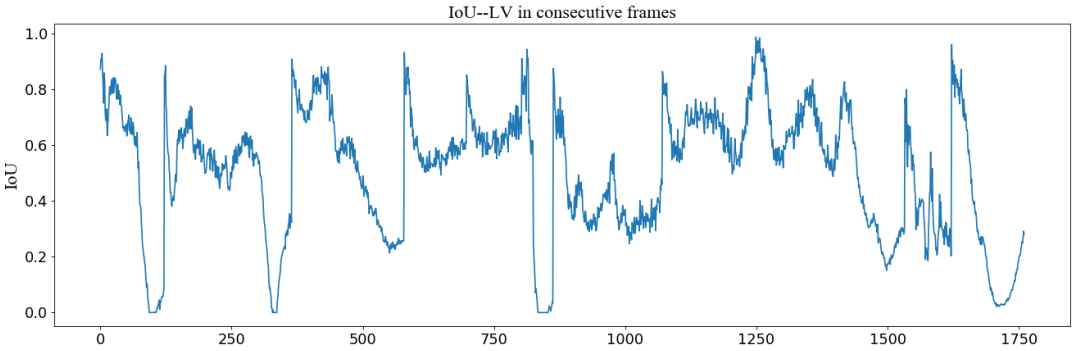

FIGURE 6.1: IoU change of leading vehicles in every two frames. The horizontal axis is the index of the frames.

Other factors could also affect the reliability of our assumption, such as the sampling frequency of our camera. Theoretically, the higher the sampling frequency is, the fewer changes the leading vehicles' bounding boxes have. Based on our experiments, if the ground truth data's sampling frequency is higher than 10 FPS, it would be enough for executing the leading vehicle detection task. Besides, the higher the

sampling frequency is, the higher the mean IoU we have.

6.2 Identifying the Leading Vehicle

Next, we need to identify, in a camera frame, which vehicle is the forward-leading vehicle. In the BDD100K dataset, the cameras are mounted on the visor of cars. Likewise, in the Viewnyx dataset, the cameras are mounted inside the windshield of trucks. For human annotators, generally, a leading vehicle is the closest vehicle in front of the camera that is driving in the same lane (including cars that are changing their lane) and roughly moving towards the same direction. In this section, we aim to solve a bi-classification task: given many bounding boxes of vehicles on an image frame, classify which one is the leading vehicle, and which ones are not.

Our first attempt is taking width, height, the x-axis, and y-axis offset of each leading vehicle as a four-dimension vector and applying a bi-classification with an AdaBoost [55] classifier. We started from an AdaBoost classifier with 10 depth-2 decision trees. As a result, the training and testing error are 8.5% and 9.2%. Since this error rate is too high, we increase the number of decision trees to 200. However, the classifier still works poorly. The training error is 6.3%, and the testing error is 6.1%. The error rate of this method is too high for us.

Our second attempt is using a geometrical constraint. As, most of the time, the leading vehicle is near the center part of our image frame, we set up a narrow vertical rectangle in the middle of the image, calculate the intersection area between the detected vehicle and the rectangle, divide it by the area of the detected vehicle, as shown in Fig. 6.2 (left). Then, we removed the division part and concise the formula to a simple x-axis coordination comparison to reduce the computation. Let the left and right x-axis coordinates of the detected vehicle be x_1 and x_2, and the left and right x-axis coordinates of the central rectangle be s_1 and s_2, as shown in Fig. 6.2 (right). The constraint of classifying leading vehicle is shown in Algorithm 1. In our case, we have 640×480 images as input and use $s_1 = 305$ and $s_2 = 335$. The overlapping threshold T ranges from 0 to 1.0, and the higher the T is, the more strict the constraint is. Here we use $T = 0.3$. Using this geometrical constraint, we have an error rate of 1.48%, which is considerably lower than the AdaBoost method. We also find that, if we let $T \leq 0.2$, the error rate will be nearly zero. Though further studies could be done on this topic, $T = 0.3$ is well enough for us. Therefore, in this chapter, we use this simple geometrical method to classify whether a detected vehicle is a leading vehicle.

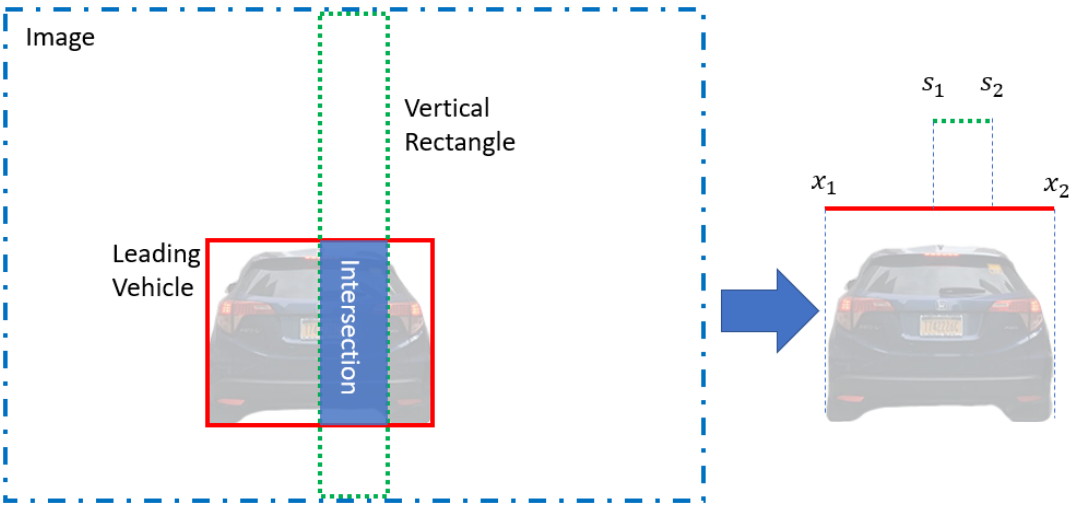

FIGURE 6.2: Geometrical constraint for identifying the forward-leading vehicle.
The left part shows the initial geometrical constraint; the right part shows the simplified constraint.

Algorithm 1: Classifying leading and sideways vehicles.

input : x_1, x_2, s_1, s_2, and a threshold T

```
if if x1 > s1 and x2 < s2 then
    return Leading ;
else if x2 − s1 > T × (s2 − s1) and x1 − s2 < −T × (s2 − s1) then
    return Leading ;
else
    return NotLeading ;
end
```

6.3 Fine-tuning the Detection Models

Since the filtered BDD100K dataset has a considerably larger amount of data than the Viewnyx dataset, and these two datasets are all about vehicle detection, we chose to train our models on the filtered BDD100K dataset and use the Viewnyx dataset for *fine-tuning* only. Fine-tuning is a technique of tweaking a model that is trained for a specific task and making it fit with another similar one. The models we have in Chapter 5 are trained on the filtered BDD100K dataset, whereas our application is based on the Viewnyx dataset. In these two tasks, they share the same objective of vehicle detection, but the quality of the pictures and the vehicles' feature in these two datasets are not the same. Therefore, in Chapter 3, we cropped and resized the images and annotations in the BDD100K dataset to make it resemble the Viewnyx dataset more, such that detecting vehicles in these two datasets are two highly similar tasks.

We fine-tune our SSD-LW-KM model, a well-performed detector on the filtered BDD100K dataset with our Viewnyx dataset. During the fine-tuning, we freeze all

the layers in the pre-trained model except the last output layer, which is a 1×1280 array. Then we train the unfrozen layer with our Viewnyx dataset.

6.4 Combine Detection and Tracking

We aim to find a method that can detect a single object with some given information from previous image frames, and it should be a light-computation method. *Object tracking* suits our needs perfectly. Therefore, in this section, our work is to combine detection and tracking, to get higher frame rate on forward-leading vehicle detection.

6.4.1 System Structure

Fig. 6.3 shows the system structure. The detection branch is using our SSD-LW-KM model with a 300×300 input size. There are many choices when selecting object trackers for the tracking branch, which will be discussed later in the next subsection. First, we set up the maximum number of consecutive tracking frames, N, after each detection. For each input frame, if N tracking frames have been processed, the detection branch is used, whether a leading vehicle is detected in the previous frame or not, to refresh the accumulated error produced by tracking. If $i < N$, the tracking is used when we have a prediction result in the previous frame, and the detection is used when we do not have it.

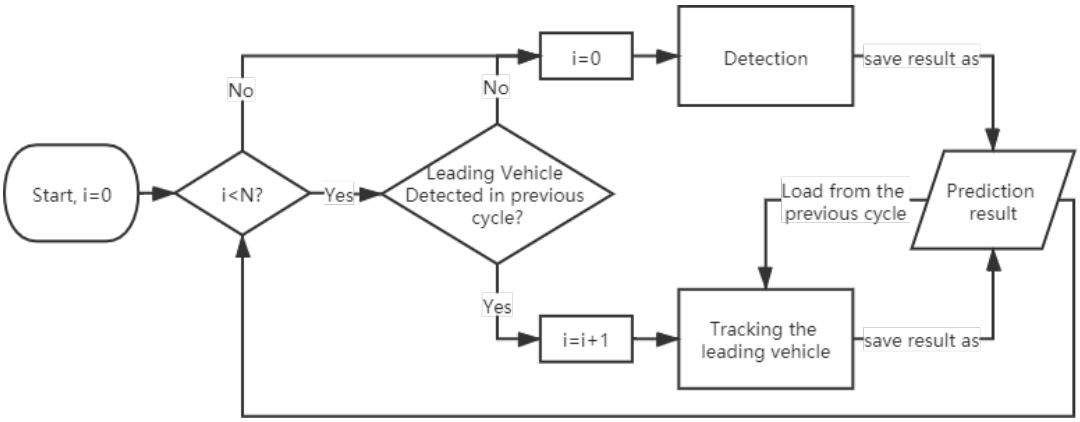

FIGURE 6.3: Combining detection and tracking.

Our system is designed for using a single core in a multi-core CPU because, in our project, there will be multiple applications running on the same mobile device, competing for computing resources. If the system can be run on an idle multi-core CPU, the system structure should be altered slightly. Multithreading and multi-core CPU task scheduling could be used for getting higher frame rate in this case.

6.4.2 Performance of Different Trackers

In object tracking, there are multi-object tracking methods and single-object tracking methods. Since, in our scenario, only one leading vehicle needs to be detected, we use single-object tracking methods. We compare multiple frequently used single-object tracking algorithms on our example videos. The involved object trackers are mentioned in Section 2.5, including *MIL*, *KCF*, *TLD*, *MedianFlow*, *MOSSE*, and *CSRT*. Though the *GOTURN* [21] tracker is known as having better performance than the other trackers above, it is not considered because it is a deep learning approach that requires relatively high computation capability and large memory size.

The experiment of comparing object trackers has two steps. The first step is editing example videos. We extracted ten short video sequences from our Viewnyx dataset and the annotation of the leading vehicles in every frame. The videos have different numbers of frames, ranging from 73 to 124. We kept the original resolution of videos, which is 640×480 pixels. In each video sequence, the leading vehicles are always the same vehicle as in the first frame. It is for having easier statistical and visual analysis. The second step is tracking the leading vehicle that appeared in the first frame, with all the object trackers we have.

We evaluate the experimental results in three aspects. The first is *Robustness*, which is the number of frames the tracking persists without losing the targeted vehicle divided by the total number of frames. The second one is *Accuracy*, which is the average of per-frame overlaps (here we use mean IoU to show the overlapping) between the tracked leading vehicle and the ground truth. The last one is the frame rate on a CPU. In the end, we check and analysis the tracked bounding boxes visually.

The experiment is executed on an i7-7700HQ CPU, and the result is shown in Table 6.1. If the robustness reaches 1.0, it means in each frame, there is a tracking result of the leading vehicle, and the IoU between the tracked and ground truth box is greater than 0. The MedianFlow tracker has the highest accuracy and very high frame rate on the CPU, making it the best candidate for accelerating vehicle detection. The TLD tracker is our second-best candidate.

Next, we give a set of results on an example video, visually presenting the tracking results to ensure the tracker is feasible for practical vehicle detection. Among the tracking results, we pick the 1^{st}, 20^{th}, 40^{th}, 60^{th}, and the 80^{th} frame, draw the ground truth leading vehicles in green, and the tracked leading vehicles in red. For readability, instead of presenting the results of all of these six methods, we present the result of TLD and MedianFlow only, shown in Fig. 6.4. Most of the time, the MedianFlow tracker produces higher IoU of leading vehicles and works much faster than

Tracker	Robustness	Accuracy	CPU Frame Rate (FPS)
KCF	0.19	0.1221	1045
MOSSE	0.17	0.1159	7748
CSRT	1.0	0.2335	42.46
MIL	1.0	0.2309	16.52
TLD	1.0	0.5233	17.64
MedianFlow	1.0	0.7472	213.1

TABLE 6.1: Comparison of off-the-shelf single-object tracking methods.

the TLD.

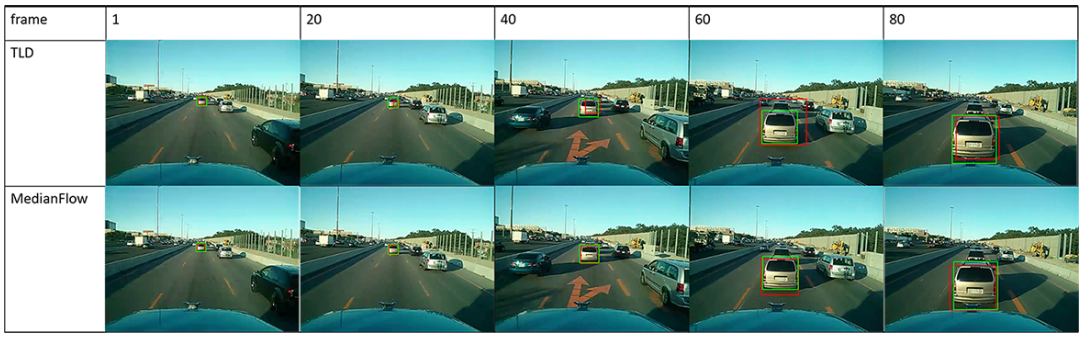

FIGURE 6.4: Tracking Result of the TLD and MedianFlow on Leading Vehicle.

Based on the experiments above, we choose the MedianFlow tracker and apply it to our system's tracking branch. As our system contains an SSD-LW-KM detection branch and a tracking branch with the MedianFlow tracker, for a more comfortable expression, we name this method as *detection-tracking* scheme.

6.5 Experiment Results

The data used in this chapter are the videos from the Viewnyx dataset. As a recap, the Viewnyx dataset contains 23 video sequences. We randomly split the video sequences into a training set containing 10 sequences, a validation set with 2 sequences, and a testing set including 11 sequences. The training set and the validation set are for fine-tuning our SSD-LW-KM models, while the testing set is for evaluating our detection-tracking scheme.

6.5.1 Evaluation

Evaluation Technique

As a recap, in our detection-tracking scheme, when there is a frame that has a successful detection of a leading vehicle, we track the vehicle for at most N frames. Our detection-tracking scheme that has at most N tracking frames will be presented as $D + TN$, where $N \in \{5, 10, 15, 20, 50\}$. Then, we have a pure detection system that uses our SSD-LW-KM model, and this system will be presented as D. In each testing, we record the *time-to-completion (TTC)* of all the video sequences, compute the equivalent frame rate in FPS, and the distribution of the IoU between the detected/tracked leading vehicle and the ground truth boxes. If there is a ground truth leading vehicle in a frame while the system failed to detect it, the IoU is 0. More than that, we also compute the width prediction error of the leading vehicles, which will be used in our next chapter, for distance estimation.

IoU

Fig. 6.5 shows the IoU distribution of the detection (D) and detection-tracking ($D + TN$) methods. As expected, the pure detection produces the highest mIoU of 0.7622. When increasing the maximum number of tracking frames N, the mIoU decreases gradually, but to a minimal extent. Even with $N = 50$, the mIoU is 0.7356, which is only 2.66% less than the pure detection. We use three commonly used metrics in statistics for evaluation, the Q1 (lower quartile, the bottom of the boxes in the figure, 25% of data is smaller than this value), Q3 (higher quartile, the top of the boxes, 25% of data is greater than this value) and *Interquartile Range (IQR)* (where $IQR = Q3 - Q1$, having a smaller IQR means the data is more stable and close to the median value).

Interestingly, from D to $D + T15$, the Q1 and Q3 values are almost the same, and the Q1 value of $D + T20$ is even higher than in the case of the pure detection. The IQR of $D + T20$ is 0.124, which is slightly smaller than pure detection D ($IQR = 0.131$). $D + T50$ also has a small IQR but its Q1 and Q3 are considerably smaller than pure detection D. It indicates that the robustness of the detection-tracking scheme could be the same as the detection, and sometimes more robust as long as the N is not too large.

Table 6.2 presents the mean and median IoU of the leading vehicles using our detection-tracking scheme. The $D + T5$ scheme gives higher median IoU than pure detection D.

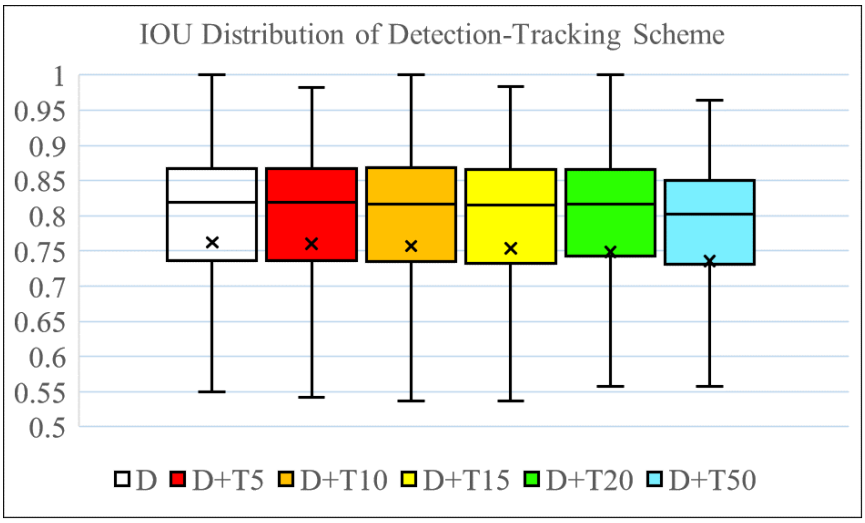

FIGURE 6.5: IoU distribution of detection-tracking scheme.

Method	Mean IoU	Median IoU
D (SSD-LW-KM)	**0.7622**	0.8192
D+T5	0.7599	**0.8194**
D+T10	0.7567	0.8169
D+T15	0.7536	0.8156
D+T20	0.7486	0.8170
D+T50	0.7356	0.8027

TABLE 6.2: IoU result of detection-tracking scheme.

Frame Rate

To evaluate our methods' CPU frame rate, we did several experiments on an idle i7-7700HQ CPU. The i7-7700HQ CPU is from a laptop installed with a CPU-version TensorFlow. Fig. 6.6 shows the results. The frame rate of $D + T5$ is roughly 3 times higher than the frame rate of the pure detection D. The $D + T15$ and $D + T20$ methods are roughly 5 times faster than pure detection on an i7-7700HQ CPU. The result shows that our detection-tracking scheme can increase the detection speed significantly.

Together with the result in the previous section, we know that our detection-tracking scheme increases the CPU frame rate by 300% to over 660%, while the largest mIoU loss is only 2.66%.

Width Prediction Error

We then evaluate width error of predictions because of the leading vehicles' width will be used for distance estimation in Chapter 7. We define a *width prediction error (WPE)* based on the the predicted width w_{pred} and width of the ground truth leading

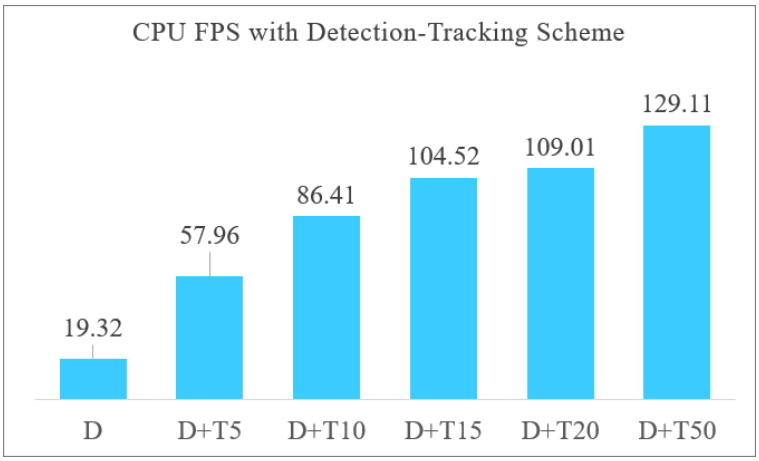

FIGURE 6.6: CPU frame rate of our detection-tracking scheme.

vehicle w_{gt}, which is given in Equation 6.1, where the *abs* means taking the absolute value of the result. Consequently, the smaller the WPE is, the more accurate the width prediction result we have.

$$WPE = abs(1 - \frac{w_{pred}}{w_{gt}}) \tag{6.1}$$

Fig. 6.7 shows the distribution of the width prediction error. In this figure, all the outliers are presented with dots. The outliers could sometimes be large if the system detects the wrong vehicle. In our experiments, the largest width prediction error is around 5.0. Therefore, the mean WPE is always considerably larger than the median WPE. Here we evaluate the performance with both the mean and median WPE. Interestingly, the lowest mean WPE does not belong to the pure detection case D but is produced by the $D + T5$ method. Then, with the increment of the maximum tracking frames N, the mean WPE goes up slightly, while the median WPE and Q1 values are roughly identical, and Q3 values fluctuate slightly. Such a result means that increasing N results in a larger error in some cases, but the overall width prediction remains acceptable. In short, the detection-tracking scheme can maintain a low WPE and sometimes provides WPE that is lower than pure detection, making the whole system stable.

6.5.2 Conclusion and Further Discussion

Based on the experimental results on IoU, CPU frame rate, and width prediction error of the leading vehicle detection, we conclude that our detection-tracking method successfully accelerated the forward-leading vehicle detection. In the contrast, the increment of the width prediction error on leading vehicles is tiny and acceptable.

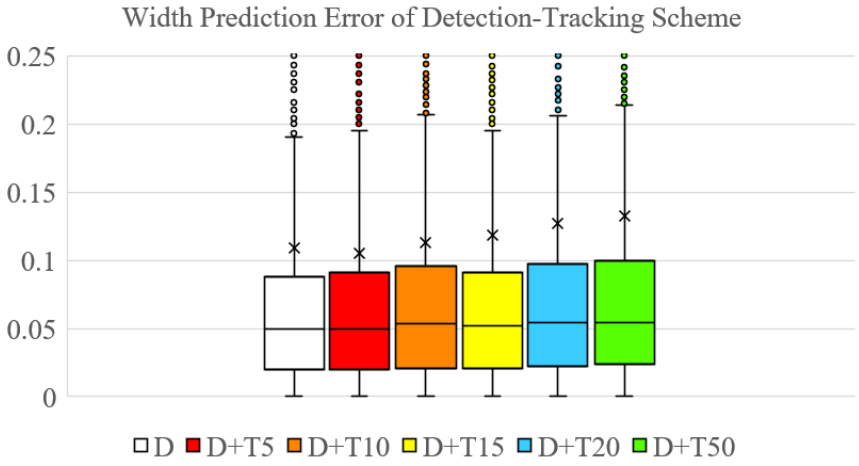

FIGURE 6.7: Width Prediction Error of Detection and Detection-Tracking methods.

There are some certain limitations in our detection-tracking scheme. First, the performance of our method depends highly on the sampling frequency of the input image frames. If the sampling frequency is high enough, for example, more than 30 FPS, the tracking branch could be appropriately applied, and the whole system could run faster but still maintain decent performance. On the contrary, if the sampling frequency is low, for example, lower than 5 fps, the tracking branch might lose tracking the vehicle, and the detection branch gets activated all the time. Second, in our work, the forward-leading vehicles are defined by the overlapping area between the detected vehicles and a simple geometric boundary in the image frame. For designing a better definition of leading vehicles, many other factors need to be considered, such as the vehicles' orientation and the way we mount our cameras, including offsets, pitch, yaw, and roll angles. Using lane detection to find forward-leading vehicle could be a good choice if computation is not a major concern. For the moment, our definition forward-leading vehicle is well enough for the later works.

Chapter 7

Forward Collision Warning System

Forward Collision Warning (FCW) systems are standard modules used in modern Advanced Driver Assistance System (ADAS), which warn drivers about the imminent collision with a forward-leading vehicle. When a FCW system-equipped vehicle comes too close to another vehicle in front of it, the FCW systems should be able to trigger visual, audible, or tactile signal to alert the drivers. More than that, some advanced FCW systems also offer autonomous braking functions if the driver remains unresponsive after the alert.

In identifying whether there is a potential danger of collision, it is crucial to judge the distance between the leading vehicle and the car itself. There are multiple types of sensors that are suitable for such a distance estimation task, such as LIDAR, Radar, and stereo cameras. We try to find an economical approach in our application by implementing an FCW system on smartphones with its primary camera. Our smartphone's primary camera is a monocular RGB camera that cannot provide depth/distance data directly. Thus, we have to estimate the distance strictly from visual information.

Chapter 6 presented a fast forward-leading vehicle detection system that can predict the bounding boxes of the vehicles. This chapter aims to estimate the distance using the bounding boxes and then build a simple FCW system.

We present the workflow of our FCW system in Section 7.1. Next, we explain our distance estimation and collision warning techniques in Section 7.2 and Section 7.3. Then we present two of our implementations developed for Python and Android devices.

7.1 Workflow of FCW system

Fig. 7.1 shows our FCW system's workflow. First, video frames are loaded from the primary camera of our smartphone. For each image frame, the SSD-LW-KM object detector and our Detection-Tracking scheme is applied to get the prediction of the

forward-leading vehicle. We estimate the distance of the vehicle based on the detected bounding box. Then, we analyze the danger level of the potential collisions based on the estimated distance. The input image, the bounding box, the estimated distance, and visual alert are presented on the smallphone screen. All the data is also compressed and saved in an alerting log. In this chapter, plotting and logging details are not included.

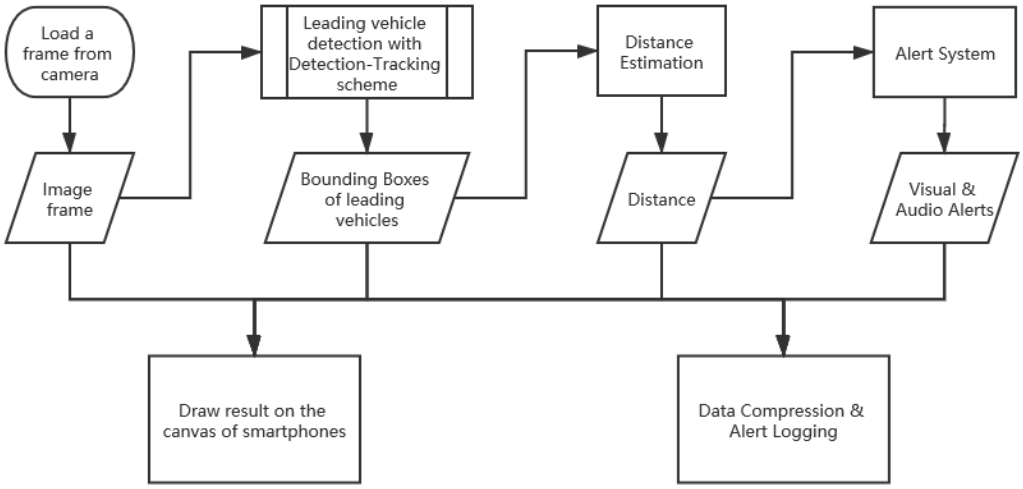

FIGURE 7.1: Workflow of our FCW system.

7.2 Distance Estimation

Our goal is to find a distance estimation method for monocular cameras with low computational complexity. Also, since we use the camera of the mobile phone mounted on the sun visor, this method should not be affected by the camera's installation angle.

7.2.1 Challenges in Directly Solving the Distance

We tried to solve the distance directly but found it is mathematically challenging.

Our goal is to compute the distance d_t between the camera and the leading vehicle at time t. Let $\vec{v}_t = [w_t, h_t, cx_t, cy_t]$ be the width/height and x/y-axis offset vector of the detected forward-leading vehicle at time t. Since the image frames are sampled at a certain sample rate, for each video sequence, w_t, h_t, cx_t, and cy_t are values from four unknown discrete functions $w(t), h(t), cx(t)$ and $cy(t)$. As we have multiple video sequences, we have four sets of functions. In linear algebra, the relationship between distance d and the detected forward-leading vehicle could be

seen as a non-linear functional that mapping from a vector space V to the distance d, where all the vectors $\vec{v} \in V$ can be expressed as $\vec{v} = [w, h, cx, cy]$. Why it is a functional but not a regular function? The answer is, there is a set of functions that have the same properties as vectors (isomorphic to vector v) and produces different real scalar, *i.e.* the distance d. For example, given a specific bounding box in a frame at time t (vector v_t), we can get various distance d_t because of many reasons, such as the different vehicle sizes/locations and different camera angles. The functional is non-linear because our camera is from smartphones and is not a typical pin-hole camera.

Generally, to compute the distance, we need to acquire the non-linear functional first according to the previous frames' values. Here we consider the most simple and straightforward situation that the non-linear functional is a polynomial of four sets of functions of $w(t), h(t), cx(t)$, and $cy(t)$, and we have sufficient bounding boxes from previous frames. However, even under such a straightforward and ideal situation, there are still three challenges when solving the functional. The first challenge is that, for each leading vehicle, we need to solve its functional, which needs vehicle identification then require higher computation. The second challenge is how to find the highest order in the functional. The third challenge is that the functional's highest order is affected by the sample rate of the videos. Furthermore, if the non-linear functional is not a polynomial, it would be harder to compute. As a result, it is very challenging, time-consuming, and computation-consuming to solve every leading vehicle's functional. Hence, for our work, it is not feasible to compute the distance d_t directly.

Approximating the w_t, h_t, cx_t, and cy_t is easier than computing the functional of d_t, and the computation growth is negligible. Therefore, we propose to break down the distance estimation task into three steps: The first step is predicting the $\vec{v_t} = [w_t, h_t, cx_t, cy_t]$, which is done in Chapter 6; The second step is smartphone camera calibration; The third step is selecting useful dimension(s) in $\vec{v_t}$ and creating a projection curve from that/those dimension(s) to $d(t)$.

7.2.2 Smartphone Camera Calibration

The unit of distance d is in meters while the unit of $\vec{v}$ is in pixels. Our aim is not to compute distance d in the camera coordinate system, but d_W, the distance of the leading vehicle in the world coordinate system. Therefore, we need to transform $\vec{v}$ on the image plane, into $\vec{v_W}$ in the world coordinate system. Equation 7.1 shows a standard 3D transformation matrix. In our case, the sub-matrix S always contain zeros only, and the sub-matrix I is always $[1]$ since we are not interested in distort the projection of any objects. Given a point $P_W = [x_W, y_W, z_W]^T$ in the world coordinate system, the corresponding projection on the image plane would

be $P = [x, y]^T = \mathcal{F}_{\mathcal{W}}(R \cdot P_W + T, K)$, where the R and T denote the rotation and translation matrices from the world coordinate system W, to the camera coordinate system, K is the intrinsic matrix of the camera, and $\mathcal{F}_{\mathcal{W}}$ is a camera model function. With $\mathcal{F}_{\mathcal{W}}$, we can plot projection curves from $\{w, h, cx, cy\}$ to their corresponding values in the world coordinate system.

$$Q = \begin{bmatrix} R & T \\ S & I \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} & \begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 \end{bmatrix} \end{bmatrix} \tag{7.1}$$

Equation 7.2 shows the definition of intrinsic matrix K.

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

According to the official specification document[1], our phone (the LG Nexus 4 that used when acquiring the Viewnyx dataset) uses a SONY IMX111PQ camera as its primary camera, whose specification is provided in Table 7.1.

Specs of the primary camera	
Focal length	$f = 3.51mm$
Sensor size	$S_w = 3.67mm$, $S_h = 2.76mm$
f-number	f/2.4
Crop factor	9.42
Resolution	640×480
Pixel size	$0.005734mm$(w), $0.00575mm$(h)

TABLE 7.1: Specification of our smartphone camera.

In the beginning, we assume that our smartphone camera is a pinhole camera, which gives the expected intrinsic matrix K_e shown in Equation 7.3. However, after some experiments on camera calibration, we found that the camera's focal length f is changing because of an embedded autofocus mechanism. There is a way to alleviate this problem once and for all–disable the autofocus function and use a fixed focus. However, such a method might hurt our vehicle detector's performance since all the images in the Viewnyx dataset are obtained with the autofocus mechanism turned on, and disabling the autofocus reduces the sharpness of the images. We tried to investigate what kind of autofocus mechanism does our camera has. Unfortunately, the datasheet of the IMX111PQ camera has been removed from the official site, and it is not possible to find the autofocus mechanism. Consequently, we have to disable

[1] Specs of our phone: https://www.devicespecifications.com/en/model/73c42796

the autofocus, and use the fixed focus for distance estimation.

$$K_e = \begin{bmatrix} 612.09 & 0 & 320 \\ 0 & 610.43 & 240 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.3}$$

Considering that the parameters of each batch of cameras from the factory are slightly different from the values in the datasheet, and the autofocus is disabled, instead of using K_e, we use the classical checkerboard pattern and the OpenCV toolbox for camera calibration. The calculated intrinsic matrix K is shown in Equation 7.4, where the f_x and f_y are smaller than the value from the official specification.

$$K = \begin{bmatrix} 575.8893 & 0 & 320.1505 \\ 0 & 578.8693 & 240.1275 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.4}$$

7.2.3 Dimension selection: Width

Here we discuss which dimension(s) in $\{w, h, cx, cy\}$ is(are) useful for distance estimation.

In Section 6.1, our first assumption is that the orientations of the leading vehicle and the camera are roughly the same. As the detected leading vehicles roughly locate in the horizontal center of the images, most of the time, the x-axis offset cx is close to the center of the image. The y-axis offset cy is highly affected by the mounting height of the camera, the distance of the leading vehicle, and especially the pitch angle of the camera. Therefore, cx and cy are not beneficial for distance estimation. The vehicle height is affected by the type/model of vehicles and the mounting height of the camera. Hence, h is not suitable for distance estimation. When cx is close to the image center, the pixel width of the vehicle w is a relatively independent factor that only related to d_W and actual width of the vehicle in the world coordinate system, w_W. Though the widths of different types of vehicles are not exactly the same, it is possible to find baseline widths for the most vehicles based on statistics. When we take a small baseline width, for wider vehicles, the estimated distance will be slightly closer, which provides our system a safety buffer zone. Therefore, we use vehicle width w for leading vehicle distance estimation.

According to our experiments, the cx and cy do not have obvious impact on d compared with w. Therefore, we chose to make a projection from w to d.

7.2.4 Distance Projection Curve

In this section, we build a distance projection curve where the entry is w, the pixel width of the leading vehicle, and the output is d_W, the distance from the camera to the vehicle.

In our experiments, we use a horizontal bar to substitute for the vehicle width w, as shown in Fig. 7.2, where the O_W and O_C are the origin of the world and camera coordinate systems. The x, y, and z axes in these two coordinate systems have the same orientation. Let the two endpoints of the horizontal bar be $P_{W,1} = (\frac{w_W}{2}, y_{W,1}, z_{W,1})$ and $P_{W,2} = (-\frac{w_W}{2}, y_{W,1}, z_{W,1})$, where the w_W is a known value, the bar width. The distance between the camera and the forward-leading vehicle is given by the translation vector $T_z = (0, 0, d_W)$, and the rotation matrix R will be identical. Then, we can find P_1 and P_2, the projection of $P_{W,1}$ and $P_{W,2}$ on the image plane, by applying $P_1 = \mathcal{F}(P_{W,1} + T_z, K)$ and $P_2 = \mathcal{F}(P_{W,2} + T_z, K)$. After that, we compute the pixel distance from P_1 to P_2 using $w = ||P_1 - P_2||$.

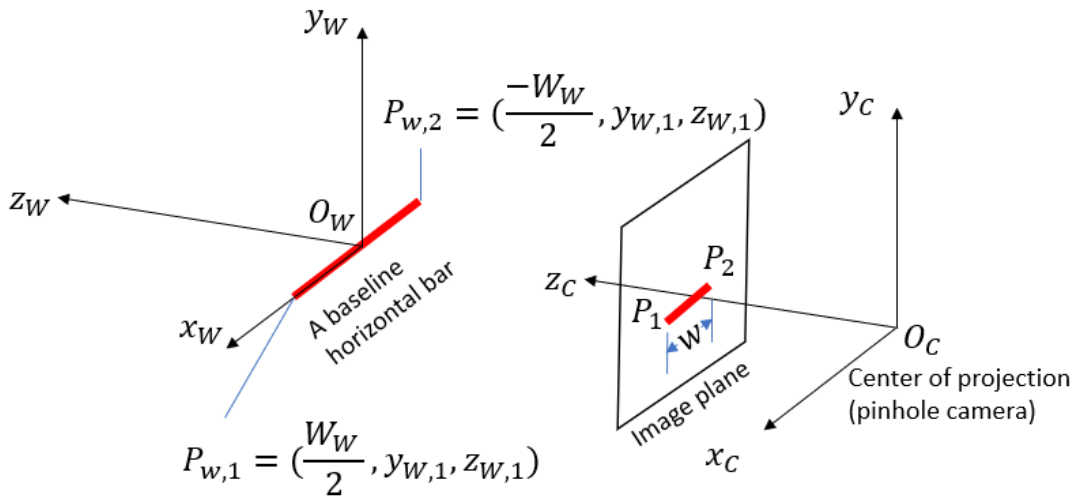

FIGURE 7.2: Coordinate systems in the horizontal bar experiment.

In the horizontal bar experiment, for each set of input $[K, w_W, d_W]$, we compute the corresponding w, the vehicle width on the image plane. We tried different values of d_W and create a lookup table, where the entry is w and the output is d_W. Since the entry of the table is discrete (in pixel width), we apply linear interpolation to the table to plot a distance projection curve, in the form of $d_W = \mathcal{F}_d(w, w_W, K)$.

To verify the accuracy of the distance estimation that uses our projection curve, we take several photos of the horizontal bar at different distances, predict the distance, and compare the result with the measured distance. As shown in Fig. 7.3, the predicted distances are only slightly different from the ground truth value. Next, we create the distance projection curve as shown in Fig. 7.4 (left), where the w_W is set to 160cm. The figure on the right side shows the estimated distance changes per pixel.

With the decrease of the width w, the distance estimation error goes up exponentially, which means that this method produces large distance estimation error when the leading vehicle is further away from the camera.

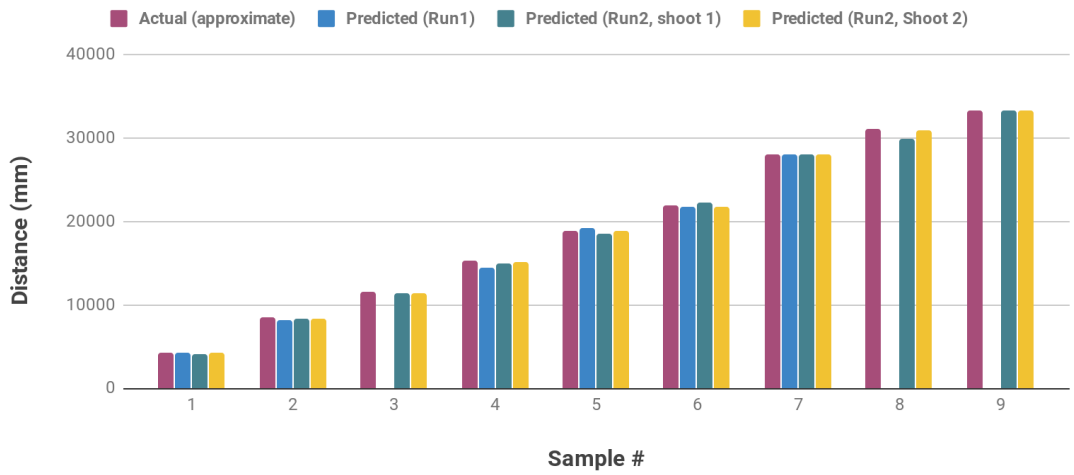

FIGURE 7.3: Verifying the distance estimation with actual distances.

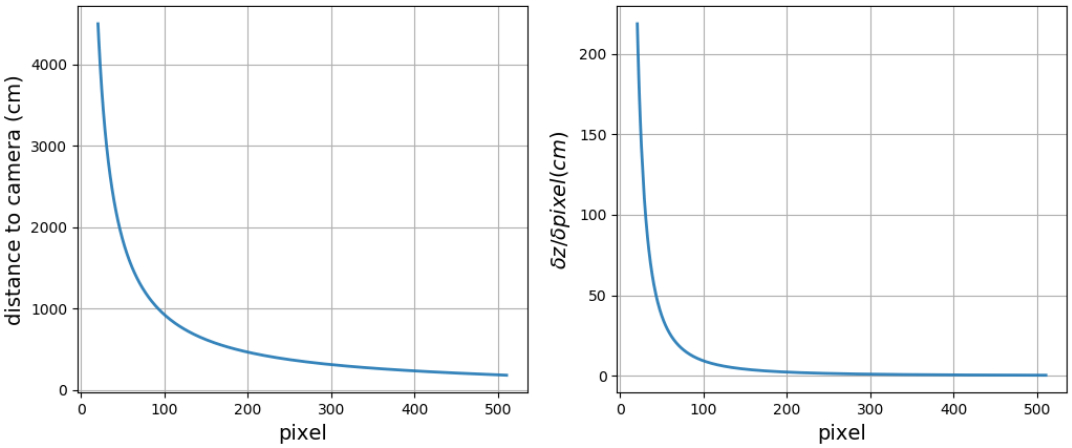

FIGURE 7.4: An example distance projection curve and the distance changes per pixel.

The bar width w_W stands for a baseline width of leading vehicles. According to an online resources[2], the majority of cars (not trucks) are between 160cm and 220cm wide. Therefore, we plot the projection curve for different values of w_W from 160cm to 220cm, with an interval of 10cm, which is shown in Fig. 7.5. Then, we can compute the distance of forward-leading vehicles with their width, and the result will be discussed in the next section.

In addition, in multi-class detection, there could be several classes of vehicles, such as cars, trucks, and buses. In this case, we have multi-class detection results,

[2] www.automobiledimension.com

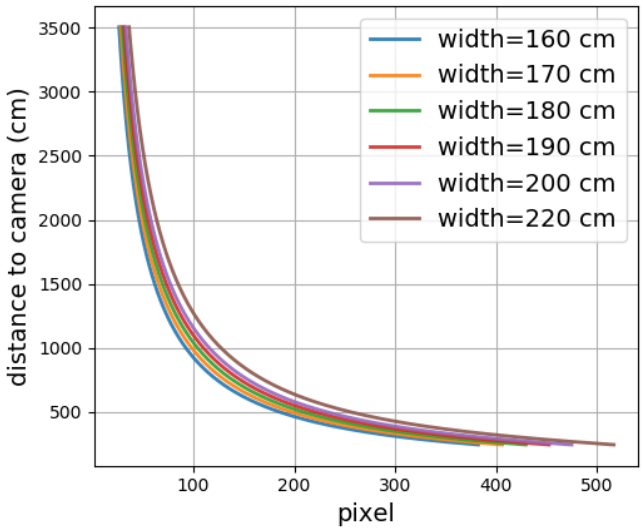

FIGURE 7.5: Distance projection curves with different baseline widths.

such that we can compute the baseline widths w_W and get multiple curves for each type of vehicle. The distance estimation could be more accurate. However, having more classes multiples the total number of predictions. Given Table 5.4 and Table 5.5, with the increment of predictions, the total processing time of the post-processing (mainly Non-Maximum Suppression) will be increased considerably. Therefore, using single-class detection is recommended unless computation is not a major concern.

7.2.5 Estimation Error

In this section, we compute the error between the estimated distance d_{est}, and the ground truth distance d_{gt}.

In the Viewnyx dataset, the ground truth distances are collected by the Radar in the *Adaptive Cruise Control (ACC)* system mounted on trucks, which is aiming to the front. The Radar system has a narrow working angle and is pointing straightly forward from the front side of the truck. Therefore, two factors make the ground truth distance noisy. First, in some cases, the Radar does not detect the forward leading vehicle but return the distance of other objects, such as the line fences on highways. Second, when the truck that holds the camera is turning, the Radar might return the distance to sideways vehicles other than the forward-leading one. Additionally, in our case, the Radar's sensitivity is 1 meter, and the distance value returned by the Radar is an 8-bit integer, outputting the distances from 1 to 255 meters. When the distance is changing but the difference is smaller than 1 meter, the Radar cannot sense the change. In our experiments, however, all the distance estimation results are using float precision, and the distances have two digits of decimals. Though the

ground truth is noisy, we did not filter it but used it directly to calculate the estimation errors.

Given the estimated distance d_{est} and the ground truth distance d_{gt}, we define the estimation error as $E_{est} = |d_{est} - d_{gt}|$, where the unit is in meters. Fig. 7.6 shows the distance estimation error of detection (D) and our detection-tracking scheme (D+T10). The left figure shows the mean estimation error of detection. When d_{gt} is greater than 20 meters, mean E_{est} are considerably higher than the situation that d_{gt} is smaller. This is mainly because of the degraded accuracy of the leading vehicle detection. The result also shows that the performance of our distance estimation method depends highly on the baseline width when $d_{gt} > 20$. We consider $160cm$ as the best baseline width because it produces the minimum distance estimation error when the range is from 0 to 20 meters, and it has a relatively small overall error. The Detection-Tracking scheme gives similar result. Compared with detection (D), our Detection-Tracking scheme (D+T10) has smaller distance errors when the distance is within 20 meters, but produces larger errors when the distance is greater than 20 meters.

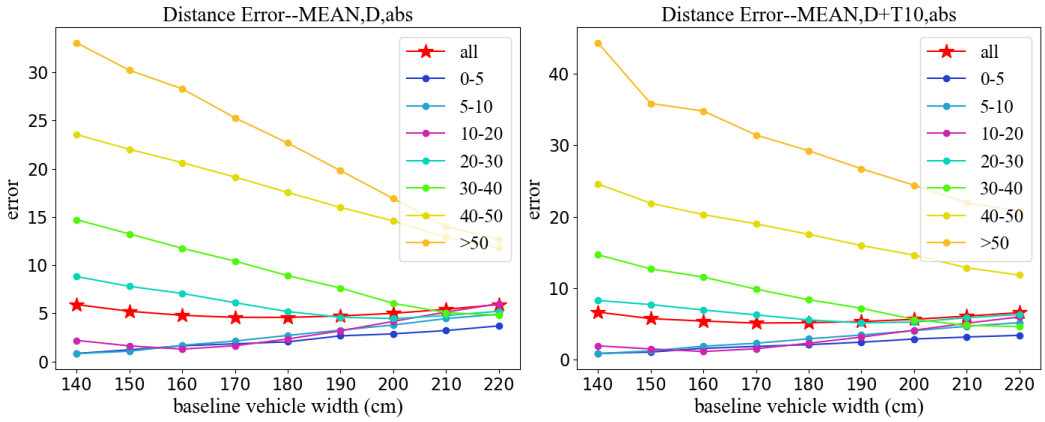

FIGURE 7.6: Mean estimation error for detection and the Detection-Tracking scheme.
The horizontal axis is baseline width w_W of the leading vehicles, and the vertical axis shows the estimation error. Each color denotes a specific distance range, *e.g.* $5-10$ means the ground truth distance is between 5 and 10 meters. The curve labeled with red stars shows the overall result that includes all the distances.

Then we compute the estimation error in percentage as $E'_{est} = \frac{|d_{est} - d_{gt}|}{d_{gt}}$, and draw the Mean Square Error (MSE) of E'_{est} in Fig. 7.7. The estimation result using $160cm$ as baseline width preserves generally a small MSE, in both the detection and the Detection-Tracking figure, which suggests that under this condition the distance estimation is more stable.

Using $160cm$ as baseline width and 0 to 20 meters as the working range, we compute the mean distance estimation error μ and the standard deviation σ, for both the

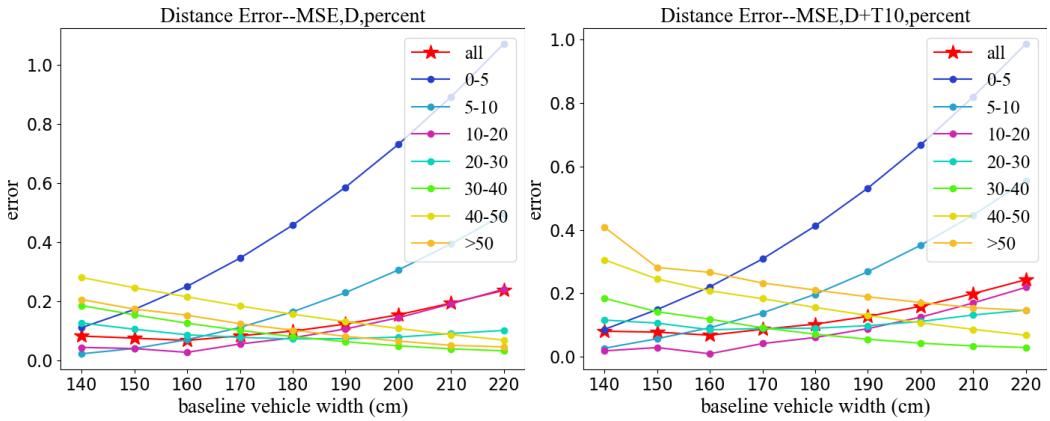

FIGURE 7.7: MSE of estimation error for detection and the Detection-Tracking scheme.

detection (D) and our Detection-Tracking scheme (D+T10). For detection, we have $\mu_D = 8.7\%$ and $\sigma_D = 5.2\%$, while the Detection-Tracking scheme has $\mu_{DT+10} = 7.9\%$ and $\sigma_{DT+10} = 9.7\%$. For a comparison, in [47], a monocular camera is used for distance estimation, with Haar-like pattern for object detection and AdaBoost for classification. It reaches $\mu = 5.0\%$ and $\sigma = 3.2\%$ for decelerating target vehicles, $\mu = 4.6\%$ and $\sigma = 2.9\%$ for stopped vehicles, and $\mu = 7.1\%$ and $\sigma = 4.5\%$ for slow-moving vehicles. As mentioned, our ground truth data is noisy and using integer precision. Therefore, our distance estimation error is large. In a practical situation, our distance estimator has a decent performance, which will be presented in Section 7.6.

7.3 Collision Warning

A practical collision warning system could be complicated since there are many factors to consider, such as the relative position, speed, acceleration of the vehicles, and different road types scenarios. In [58], a work from the Mobileye[3] company, *Responsibility-Sensitive Safety (RSS)* standard was proposed for Autonomous Vehicles to drive safely under various scenarios. In our work, we focus on the most straightforward situation in RSS, which is avoiding having collisions with forward-leading vehicles.

The safe driving distance is a fundamental factor for collision warning, which highly depends on the regulations of different countries and provinces. Appendix B shows that in Ontario, the drivers are recommended to maintain a two-second space away from the forward leading vehicle under ideal weather conditions.

[3] www.mobileye.com

In our project, we setup three danger levels from low to high. The danger levels are defined based on two factors. The first factor is the distance to the forward-leading vehicle computed from the current image frame. The second factor is the predicted distance after a minimum reaction time of the drivers. For each factor, we can compute a danger level, and the final danger level will be the higher one of the two outcomes. Note that the current danger level model is very limited and could be further customized because of many reasons, including the make and model of the car, weather, driver's habit, tire condition, etc.

7.3.1 Static Distance Thresholds

Our first type of alert is based on d_{est}, the estimated distance between the car itself and the forward-leading vehicle. The closer they are, the smaller the d_{est} is, and the more dangerous it is. Let D_1 and D_2 be two thresholds. For the situations of $d_{est} < D_1$, $D_1 \leq d_{est} < D_2$, and $d_{est} \geq D_2$, the danger level are respectively set to be *High, Mid* and *Low*, corresponding to red, yellow and green. Fig. 7.8 shows an example of a car getting close to its forward-leading vehicle. The horizontal axis is time, and the vertical axis is the relative distance to the leading vehicle. The curve is painted according to which interval the current value of d_{est} belongs to. The low danger is the normal state, while mid danger means the driver should take actions to slow down the vehicle, and high danger means a harsh stop is needed to avoid a collision. The value of D_1 and D_2 should be customized according to the make and model of the target car.

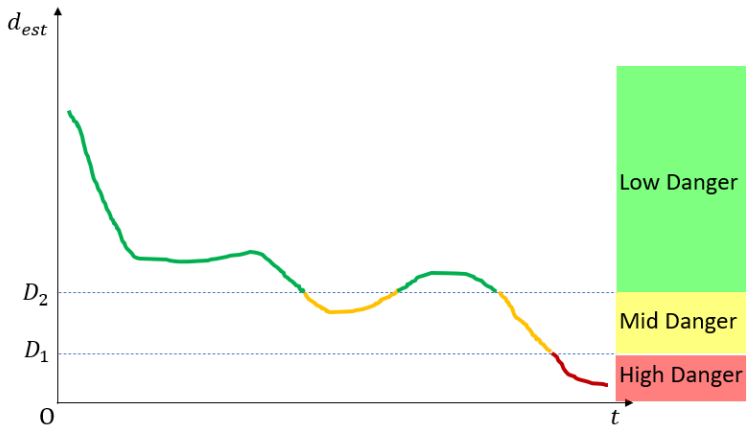

FIGURE 7.8: Define danger levels with static distance thresholds.

7.3.2 Predicted Distance after Reaction Time

The second method is based on the predicted distance d_p to the leading vehicle after t_{react}, an average reaction time for the drivers to make a harsh stop. d_{est} is still the

estimated distance between the car itself and the forward-leading vehicle.

Let the distance and time of the current input frame be d_1 and t_1. Let the distance and time of the previous input frame be d_0 and t_0. Then we compute the predicted distance d_p of the car after the needed reaction time, t_{react}. In our case, d_p is computed by a simple linear function, which is $d_p = \frac{d_1 - d_0}{t_1 - t_0} \times t_{react} + d_1$. Let $D_1 = 0$ be the threshold between high and mid danger, D_2 be the threshold between mid and low danger. Fig. 7.9. shows an example of a car getting close to its forward-leading vehicle and then getting away from it. In this example, at time t_1, we have $D_1 < d_p < D_2$, therefore the danger level at t_1 is medium and hence that part of curve is yellow. Same as the previous section, the low danger corresponds to a normal situation, mid danger suggests braking action is needed, and high danger requires a harsh braking action.

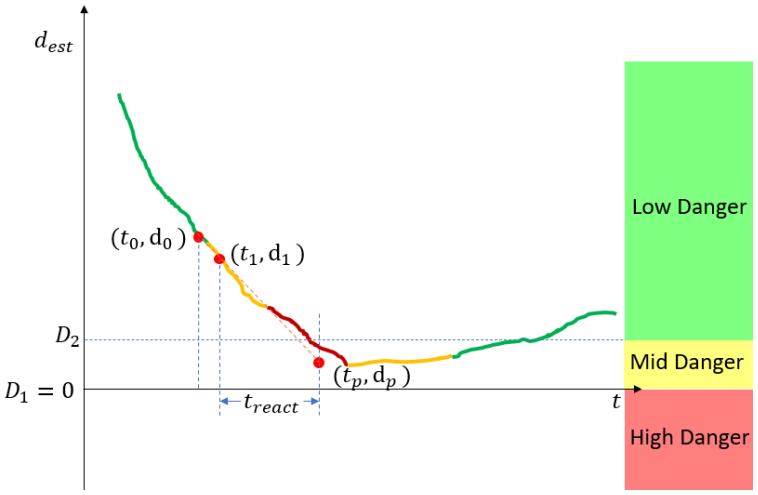

FIGURE 7.9: Define danger levels with predicted distance after reaction time.

This collision warning method could be customized according to the make/model of the cars, as well as the drivers' driving habit if possible.

7.4 FCW system for Python Environment

By combining all the previous works in this thesis, a fully functional FCW system was developed with Python, which includes leading vehicle detection, detection-tracking scheme, distance estimation, and collision warning. Since the experimental results of each part in this system are already discussed in the corresponding chapters, in this chapter we only present the performance of the whole system.

7.5 FCW system for Android Environment

We provide a partially functional Android demo to examine our detection method on a smartphone. We implement our vehicle detector with the SSD-LW-KM model to a smartphone with the Android demo provided in the TensorFlow Object Detection API[27]. The Detection-Tracking scheme is not included due to some compatibility problems. All the experiments are using detection only.

7.5.1 Specifications of Cellphone

We have two smartphones, including a HUAWEI P10 Lite, and a Google Pixel 3. Their specifications are shown in Table 7.2.

Name	Android	Processor	RAM	Resolution (resized)
HUAWEI P10 Lite	8.0	Kirin 658	4GB	640×480
Google Pixel 3	8.0	Snapdragon 845	4GB	640×480

TABLE 7.2: Specifications of smartphones.

7.5.2 Model Performances on Cellphone

In this experiment, we use our SSD-LW-KM model with an input size of 300×300 pixels. The camera frames are initially 640×480 and resized to fit the input size. All the experiments are mainly using CPU for calculation, and using GPU for drawing the results.

We record the average reference time (in milliseconds) and then compute the frame rate on CPU. The result is shown in Table 7.3. In the table, we also refer to the results in the MobileNet-V2 paper [54] for comparison. On the Kirin 658 processor, the CPU FPS of our SSD-LW-KM method is slightly faster than the SSD-Original model. Our SSD-LW-KM method reaches 6.90 FPS on a Snapdragon 845 processor. With our Detection-Tracking scheme, it is possible to increase the FPS of leading vehicle detection by 3 to 5 times and run it in real-time with a negligible decrease of accuracy.

Model	Processor of Phone	CPU FPS
MNet V1 + SSDLite [54]	Snapdragon 821	3.703
MNet V2 + SSDLite [54]	Snapdragon 821	5.0
SSD-Original	Kirin 658	1.60
SSD-LW-KM	Kirin 658	1.72
SSD-LW-KM	Snapdragon 845	6.90

TABLE 7.3: Detection FPS on smartphones.

To check the memory usage of our application, we record the result in the Memory Profiler in the Android Studio. The result is shown in Table 7.4. As a result, our SSD-LW-KM method has approximately 50% lower total memory usage than the original model. Besides, the SSD-original model uses 18659KB ROM, while our model uses a similar size of ROM of 18518KB.

Model	J/K	Native	Graphics	Stack	Code	Others	Total
SSD-Original	26.3	112.3	21.8	0.1	12.9	1.0	174.4
SSD-LW-KM	6.1	48	20.4	0.0	12.4	1.1	88.0

TABLE 7.4: Memory allocation of difference methods, all units are MB.
The *J/K* column shows the allocated memory from Java or Kotlin objects, where the Kotlin is another programming language that is designed to inter-operate fully with Java. The *Native* column shows the memory allocated by C or C++ objects. The *Graphics* and *Code* columns show the memory size allocated for processing images, flushing the screen and processing compiled codes. Note that the graphic memory is shared with GPU applications. The *Stack* is mainly related to multithreading, which is not used in our works, therefore their value is small.

In Table 3.3, we mentioned the battery consumption speed of the original SSD MobileNet-V2 models with different input sizes. Here we have the same experiment for our SSD-LW-KM model and the whole FCW system. We clear other active applications of our HUAWEI P10 Lite smartphone, charge the battery to 90%, run our application, and keep it running until the battery of the phone reaches 80%, record the time consumed. After 6 rounds of experiments, the average battery consumption speed is calculated, and the result is shown in Table 7.5. It turns out that our SSD-LW-KM model is relatively more power-saving, and the battery consumption speed of the whole FCW system is lower than the original SSD MobileNet-V2 model.

Model Name	Battery Consumption (%/min)
SSD-Original	0.55
SSD-LW-KM	0.416
Our FCW system	0.434

TABLE 7.5: Battery Drain Effect Comparison.

7.6 Performance of Practical Testing

In this section, we present some results of our practical testings.

7.6.1 FCW System in Python Environment

The first practical testing is executed with our fully functional FCW system that is built for Python environment. The whole system includes leading vehicle detection, detection-tracking scheme, distance estimation, and visual collision warning. In this experiment, 11 video sequences from the Viewnyx dataset are used, which contain 1815 frames and include various light/weather conditions. Here we only provide some keyframes as examples. A link of a demo video of this work is also provided in Appendix A.

The result of our FCW system onn a testing sequence is shown in Fig.7.10. The red bounding boxes denote the forward leading vehicles. The detection results of all the other vehicles are saved into a working log but not drawn on the screen. At the bottom left corner of each image, it shows the estimated distance to the leading vehicle, as well as the danger level for collision warning.

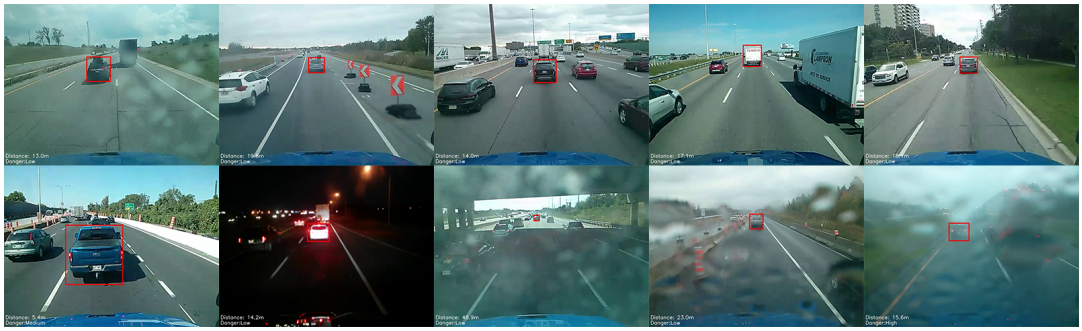

FIGURE 7.10: A few examples of our FCW system in Python environment.

Then we discuss the reliability of our FCW system. We take 11 videos in our Viewnyx dataset as input and run our FCW system. If we have a ground truth forward-leading vehicle but no forward-leading vehicle is detected, we consider that frame a failure. If there is no ground truth vehicle but we have a prediction, it is also a failure. If we have a ground truth vehicle and a vehicle is detected, but the IoU between those two vehicles' bounding boxes is smaller than 0.5, we take it as a failure. We present some failure cases of the whole FCW system in Fig.7.11. We count the failure cases manually among the 11 videos, and the result is shown in Table 7.6. We have 67 failure cases among 1815 frames, giving the *Failure Frequency* of our FCW system is 3.69%. Note that the Failure Frequency is not *Failure Probability*, but it still shows our FCW system has acceptable reliability.

FIGURE 7.11: Failure cases of our FCW system in Python environment.

Video Index	Failure Cases	Main Reason
3610	8	changing lane
3611	24	turning + changing lane
3613	0	-
3614	0	-
3617	0	-
3624	27	heavy rain
3626	0	-
3628	0	-
3632	0	-
3649	8	long distance
3652	0	-

TABLE 7.6: Failure cases per video.

7.6.2 FCW System in Android Environment

The second practical testing is done with a partially functional FCW system that is built for Android environment. In this system, the detection-tracking scheme is excluded. We use a HUAWEI P10 Lite smartphone for this experiment. The smartphone is mounted behind the front windshield of an SUV. All the videos are shot in Ottawa, Ontario.

A few examples of the FCW system is shown in Fig.7.12, in which the distance estimation is also included. In the figure, red boxes are the predicted leading vehicle, green boxes denote other vehicles detected. The brighter the red and green colors are, the higher the confidence score they have. The distance value at the bottom left of the canvas shows the estimated distance of the leading vehicle.

In our experiments, we have 3 long videos which are over 1 hour 20 minutes in total. Most of the failure cases happen at sharp turning, crossroads, and T intersection because we are using a center strip for identifying the leading vehicle and a baseline width for distance estimation. In Fig.7.13 we provide some examples.

FIGURE 7.12: A few examples of our FCW system in Android environment.

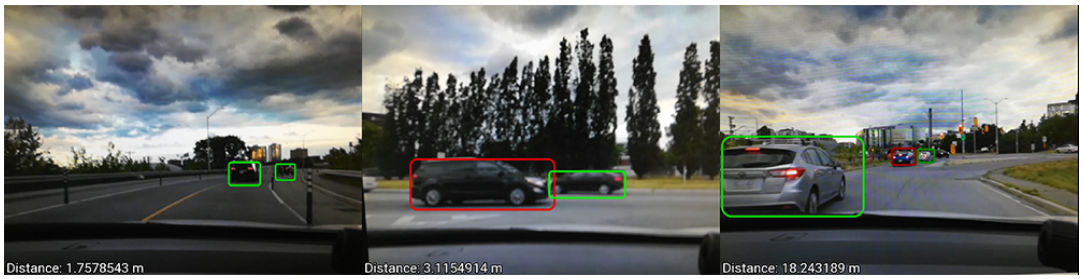

FIGURE 7.13: Failure cases of our FCW system in Android environment

Chapter 8

Conclusion and Future Works

8.1 Conclusion

In this research, we aimed to build a Forward Collision Warning (FCW) system with a low computational cost for running on a smartphone. The smartphone is installed inside the windshield of a car/truck, with its primary camera pointing forward to the road. We built this FCW system based on a forward-leading vehicle detection module.

In Chapter 3, **Dataset Analysis and Architecture Selection**, we processed and analyzed our datasets, and selected a proper object detection architecture for our FCW system. We first annotated our collected dataset with bounding boxes. Then, we pre-processed the BDD100K [65] dataset, including reshaping the images/annotations to fit the resolution of the smartphone camera and removing tiny objects that are not relevantn to our FCW system. By analyzing the bounding box shape (width/height and aspect ratio) distribution, we found box shape that could be used as a feature of objects when designing object detectors. Then, we compared multiple mobile device running object detectors and measured their memory/model size, Frame Rate, mean Average Precision (mAP), and battery drain effect. We selected the Single Shot MultiBox Detector (SSD) [41] with MobileNet-V2 [54] backbone as our base model to work on.

In Chapter 4, **SSD Customization for Vehicle Detection**, we customized SSD multi-scale feature maps and we tested two approaches to optimize the SSD default boxes. The first method considered the baseline size and aspect ratios of default boxes separately by applying Gaussian distribution fitting to the ground-truth boxes. In the second method, we optimized the shapes and the number of default boxes by applying layer-wise grouping and K-Means clustering to the ground-truth boxes.

In Chapter 5, **Experimental Results of Vehicle Detection**, we compared the performance of our SSD models in Chapter 4 in terms of Precision-Recall curve, Average Precision of different vehicle sizes, and Frame Rate on GPU/CPU. Compared with

the original SSD model, our most successful SSD model, the SSD-LW-KM model, has a better precision-recall curve, 5.23% higher AP-M (Average Precision on Medium-size objects), 4.46% higher overall Average Precision, and meanwhile, slightly higher Frame Rate. Besides, our method using Gaussian Distribution fitting, the SSD-GPDF model, gives the highest AP-L (Average Precision on large objects).

In Chapter 6, **Forward-leading Vehicle Detection Acceleration**, we combined vehicle detection and single-object tracking to boost the detection. With such a detection-tracking scheme, our detection system achieves 300% to 660% faster CPU frame rate with only 2.66% mean IoU loss, compared with using pure detection. Our D+T5 model (tracking at most 5 frames after per detection frame) provides an even higher median IoU than detection, indicating that our method is faster and more robust than the original SSD detector.

In Chapter 7, **Forward Collision Warning System**, we built a Forward Collision Warning (FCW) system, using our most successful model from Chapter 4, the SSD-LW-KM model, the detection-tracking scheme defined in Chapter 6, together with a simple distance estimation method that only needs a monocular camera. We implemented our FCW system in both Python and Android environments. In the Android environment, compared with the original SSD model, our SSD-LW-KM model requires less memory and consumes battery slightly slower. With a Snapdragon 845 phone processor, our SSD-LW-KM model (detection only) achieves 6.90 frame per second. With our detection-tracking scheme, the frame rate can be 3 to 5 times faster and achieve a real-time level.

8.2 Future Works

Object detection is a fast-evolving field, for which new techniques are proposed continuously. Several works could be done to improve our object detector, such as using better IoU calculation methods like GIoU [52], combining NMS into our detector, as proposed in [25]. When selecting the forward-leading vehicle in an image, we might be able to apply lane detection first, then find the forward-leading vehicle in that lane to avoid missing forward-leading vehicles when turning sharply. We choose to use baseline width and camera calibration to guarantee low computation for distance estimation, but it comes with a relatively high distance estimation error. A more proper way could be applying lane detection to estimate the geometrical relationship between the leading vehicle and the camera and get the distance in a bird-eye-view. In this case, we could have a lower distance estimation error.

There are also some limitations in our experiments' setup. For example, we did not include the motorcycle class because the feature of motorcycles would be significantly different than the feature of the car, truck, and bus classes. However, this

limitation could be solved if our processing unit is powerful enough to run multi-class vehicle detection. In that case, the network can learn the features of motorcycles and provide predictions of them. Then we can design distance estimation curves for them. In Chapter 7, the accuracy of distance estimation would be severely affected if the camera is mounted differently from the height and orientation in our Viewnyx dataset. Possible solutions to this problem include adding lane detection or fusing other types of sensors to help get the orientation of the camera. We could also add more nighttime images and images blurred by raindrops or mud to our dataset, to further improve our FCW system's performance in harsh weather conditions. Besides, the parameters and functions in our FCW system could be further customized based on the make and model of the vehicle that is going to equip the system, or based on drivers' driving habits, to personalize the system to achieve better performance in practical scenes.

Appendix A

Demo Videos

The first video shows a complete FCW system designed for the Viewnyx project, including leading vehicle detection, detection-tracking scheme, distance estimation, and collision warning. Check it by clicking this link: https://youtu.be/-ptvfabBZWA. Fig. A.1 shows a few examples.

FIGURE A.1: Example frames of our FCW system.

Appendix B

Reject Size of Filtering Bounding Boxes

For this vehicle detection project, one of the problems is figuring out the minimum size (in pixels) of vehicles that will be taken into our consideration. To filter the annotations that we do not need for forward leading vehicle detection, we need to figure out what is the *reject size*, which is the minimum size of the possibly dangerous vehicle that might cause collision. Thus, any vehicle smaller than that size (usually with a greater distance) is not essential for our detection system.

There are two steps before calculating the reject size: get camera specs and get vehicle specs. We extract some major specs of the primary camera (the secondary camera in the front of the phone will not be used) of LG Nexus 4 according to the list provided by Viewnyx, shown in the Table B.1.

Specs of primary camera	
Focal length	$f = 3.51mm$
Sensor size	$S_w = 3.67mm$, $S_h = 2.76mm$
f-number	f/2.4
Crop factor	9.42
Resolution	640×480
Pixel size	$0.005734mm$(w), $0.00575mm$(h)
Maximum resolution*	3264×2448

TABLE B.1: Specs of Camera from Viewnyx

Here we assume that our target is a $2m \times 2m$ square, which is an approximate size of the rear side of a car, hence we get the size of car $S_{car} = 2m$. For the safe driving distance, it varies based on different driving speed and province. We take the numbers in The Official Ministry of Transportation (MTO) Driver's Handbook as our basis. Generally, a driver should drive at least 2 seconds behind the leading vehicle during ideal conditions. In terms of the speed limit, we take the highest driving speed limit in Ontario, which is $100km/h$ ($27.78m/s$) on freeways, as our

standard. This gives us the best possible safe driving distance as follow.

$$D_{safe} = 27.78m/s \times 2s = 55.56m \tag{B.1}$$

Which means any car out of this range are not essential and could be ignored. Given the car size, safe distance and focal length of the primary camera, we can get the reject size on sensor (image plane):

$$S_{rs} = \frac{S_{car}}{D_{safe}} \times f = \frac{2m}{55.56m} \times 3.51mm = 0.1264mm \tag{B.2}$$

Thus, the reject sizes in pixels are

$$w = \frac{S_{rs}}{S_w} \times 640 = \frac{0.1264mm}{3.67mm} \times 640 = 22.04 \approx 22 \tag{B.3}$$

$$h = \frac{S_{rs}}{S_h} \times 480 = \frac{0.1264mm}{2.76mm} \times 480 = 21.98 \approx 22 \tag{B.4}$$

Therefore, the reject size of filtering bounding boxes is 22 pixels for both bounding boxes width and height.

Appendix C

Details about Aspect Ratio Customization

C.1 Low-pass Filtering

The problem of removing the pulses and valleys in the histograms is fundamentally a data smoothing problem.

In our case, the histogram could be seen as a sequence of discrete signal. We consider the vertical axis of the histogram as the signal amplitude, each bin has a sampled value from a continuous function of aspect ratio distribution. The whole histogram is then a discrete-time signal with only a single period. Hence, to smooth the histogram, we have two steps. First, we apply Fast Fourier Transformation (FFT) to get the frequency domain. Second, we set cutoff frequencies to keep the low frequencies and attenuate the frequencies higher than the cutoff frequency. We customized the cutoff frequency for the BDD100K dataset, and then repeat this step on the Caltech and Viewnyx dataset. After that, the Inverse Fast Fourier Transformation (IFFT) is applied to reacquire the signals in time-domain, which are supposed to be the smoothed histograms of aspect ratio. Note that only the BDD100K histogram needs data smoothing as it has pulses and valleys. We provide the smoothed histograms of Caltech and Viewnyx dataset, which are just for comparison. The results are in Fig. C.1. We can see that all the valleys and pulses are removed from the BDD100K histogram and hence the bias created by pre-trained object detectors is reduced. The Caltech histogram does not change much compared with the histogram in Fig. 3.4, as it does not have too many pulses and valleys before smoothing. The Viewnyx dataset is considerably smaller than the BDD100K and Caltech dataset. Therefore, it is not suitable to apply data smoothing on the Viewnyx histogram.

C.2 Gaussian Probability Distribution Fitting

In this section, to estimate the probability distribution of aspect ratios in the BDD100K dataset, we utilize Gaussian Probability Distribution Fitting (Gaussian PDF) to the

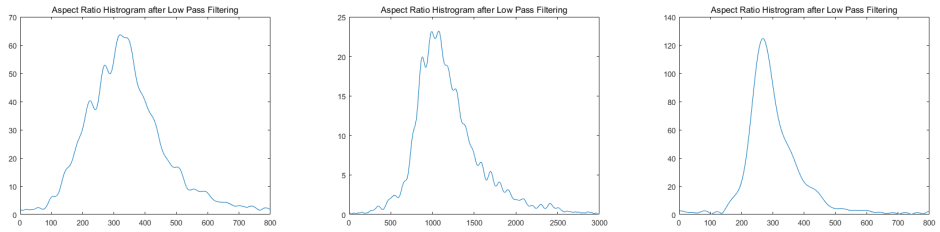

FIGURE C.1: Aspect ration distribution after applying low pass filter.
From left to right, they are of BDD100K, Viewnyx and Caltech dataset. The vertical axis is number of boxes, the horizontal axix is the "signal" length. Though all three sub-figures have signal length of 800, these three signals are corresponding to different aspect ratio ranges, which are $[0.0, 3.0]$, $[0.0, 3.0]$, and $[0.0, 1.0]$.

smoothed histogram.

The Gaussian PDF results are shown in Fig. C.2. We do not calculate the error between the Gaussian PDF result after data smoothing (the blue curve) and the ground truth aspect ratio distribution (the red histogram) because we are suspicious about the bias carried by the ground truth data. We have $\mu = 1.31$ and $\sigma = 0.66$ before data smoothing, while after data smoothing we have $\mu' = 1.28$ and $\sigma' = 0.48$. Compared to the Gaussian PDF result before smoothing (red curve), the result after smoothing (blue curve) is visually closer to both the original and smoothed histograms (red and blue histograms). Therefore, by applying data smoothing, we reduced the aspect ratio bias, and the Gaussian PDF result preserves more accurate information about aspect ratios than without data smoothing.

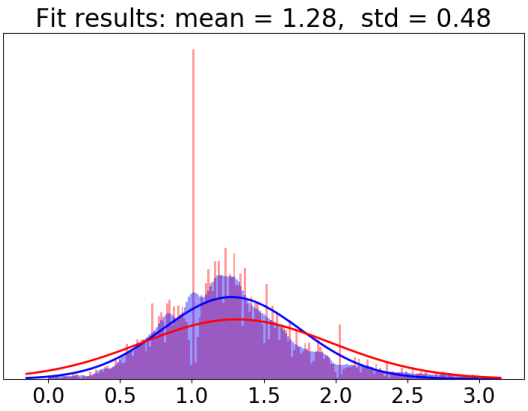

FIGURE C.2: Gaussian PDF result.
The light red and light blue histograms are aspect ratio histograms of the BDD100K dataset, before and after data smoothing. The red and blue curve denotes the Gaussian PDF result before and after data smoothing.

The aspect ratios of the default boxes are generated according to the Gaussian PDF. We replace the aspect ratios in the original model with $\{\mu' - 2\sigma', \mu' - \sigma', \mu', \mu' + \sigma', \mu' + 2\sigma'\}$, to include the majority of aspect ratios in the ground truth. Note

that the aspect ratios generated from Gaussian PDF (Fig. C.2) may not be the optimal choice, but it fits the aspect ratio distribution better than the default aspect ratios in the original SSD model. Hence, it should be helpful for the training. As a result, the customized aspect ratios are $a_{custom} \in \{0.32, 0.8, 1.28, 1.76, 2.24\}$.

Appendix D

Elbow Method

Elbow method [62] is often used to choose an optimal number of clustering center in K-Means clustering. It fundamentally takes the k value at an obvious turning point on the K-Means score curve. To achieve it, a threshold t_{elbow} is used. As shown in Fig. D.1, in our context, if $s_{5m,l}$ (the score of the l^{th} layer when $k = 5$) fulfills $s_{5m,l} > s_{4m,l} \times t_{elbow}$, we stop increasing k and keep $k = 5$ as the clustering result.

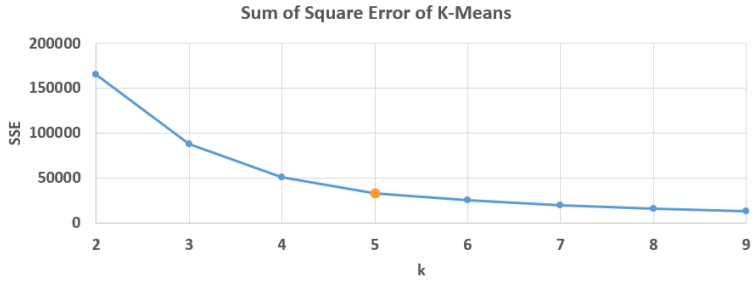

FIGURE D.1: An example of Elbow method.
The orange dot is kept as the result as it meets the requirement in Elbow method.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[2] A. A. Ali and H. A. Hussein. "Distance estimation and vehicle position detection based on monocular camera". In: *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*. 2016, pp. 1–4.

[3] "The Handbook of Brain Theory and Neural Networks". In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9. URL: http://dl.acm.org/citation.cfm?id=303568.303704.

[4] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. "Visual tracking with online multiple instance learning". In: *2009 IEEE Conference on computer vision and Pattern Recognition*. IEEE. 2009, pp. 983–990.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[6] David S Bolme et al. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2544–2550.

[7] Weifeng Chen et al. "Single-image depth perception in the wild". In: *Advances in neural information processing systems*. 2016, pp. 730–738.

[8] Jifeng Dai et al. "Deformable Convolutional Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 764–773.

[9] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1.

[10] Piotr Dollar et al. "Pedestrian Detection: An Evaluation of the State of the Art". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 34.4 (Apr. 2012), pp. 743–761. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2011.155. URL: http://dx.doi.org/10.1109/TPAMI.2011.155.

[11] Mark Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *Int. J. Comput. Vision* 111.1 (Jan. 2015), pp. 98–136. ISSN: 0920-5691. DOI: 10.1007/s11263-014-0733-5. URL: http://dx.doi.org/10.1007/s11263-014-0733-5.

[12] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[13] P. F. Felzenszwalb et al. "Object Detection with Discriminatively Trained Part-Based Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645.

[14] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[15] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[16] Huang Guan et al. "Real-time lane-vehicle detection and tracking system". In: *2016 Chinese Control and Decision Conference (CCDC)*. 2016, pp. 4438–4443. DOI: 10.1109/CCDC.2016.7531784.

[17] Muhammad Abdul Haseeb et al. "DisNet: a novel method for distance estimation from monocular camera". In: *10th Planning, Perception and Navigation for Intelligent Vehicles (PPNIV18), IROS* (2018).

[18] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[19] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[20] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[21] David Held, Sebastian Thrun, and Silvio Savarese. "Learning to track at 100 fps with deep regression networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 749–765.

[22] João F Henriques et al. "Exploiting the circulant structure of tracking-by-detection with kernels". In: *European conference on computer vision*. Springer. 2012, pp. 702–715.

[23] João F Henriques et al. "High-speed tracking with kernelized correlation filters". In: *IEEE transactions on pattern analysis and machine intelligence* 37.3 (2014), pp. 583–596.

[24] Geoffrey Hinton. *rmsprop: A mini-batch version of rprop*. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[25] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. "Learning non-maximum suppression". In: *CoRR* abs/1705.02950 (2017). arXiv: 1705.02950. URL: http://arxiv.org/abs/1705.02950.

[26] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[27] Jonathan Huang et al. "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.351. URL: http://dx.doi.org/10.1109/CVPR.2017.351.

[28] A.G. Ivakhnenko and V.G. Lapa. *Cybernetic Predicting Devices*. JPRS 37, 803. Purdue University School of Electrical Engineering, 1966. URL: https://books.google.ca/books?id=l38DHQAACAAJ.

[29] Pengchong Jin, Vivek Rathod, and Xiangxin Zhu. "Pooling pyramid network for object detection". In: *arXiv preprint arXiv:1807.03284* (2018).

[30] Apoorva Joglekar et al. "Depth estimation using monocular camera". In: *International journal of computer science and information technologies* 2.4 (2011), pp. 1758–1763.

[31] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Forward-backward error: Automatic detection of tracking failures". In: *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, pp. 2756–2759.

[32] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Tracking-learning-detection". In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2011), pp. 1409–1422.

[33] Nitish Shirish Keskar and Richard Socher. "Improving generalization performance by switching from adam to sgd". In: *arXiv preprint arXiv:1712.07628* (2017).

[34] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization. ICLR (2015)". In: *arXiv preprint arXiv:1412.6980* 9 (2015).

[35] Robin Kreuzig, Matthias Ochs, and Rudolf Mester. "DistanceNet: Estimating Traveled Distance from Monocular Images using a Recurrent Convolutional Neural Network". In: *CVPR Workshops*. 2019.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[37] S. Kuutti et al. "A Survey of Deep Learning Applications to Autonomous Vehicle Control". In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–22. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2962338.

[38] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/iccv.2017.324. URL: http://dx.doi.org/10.1109/ICCV.2017.324.

[39] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[40] Mason Liu et al. "Looking fast and slow: Memory-guided mobile video object detection". In: *arXiv preprint arXiv:1903.10172* (2019).

[41] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[42] Alan Lukezic et al. "Discriminative correlation filter with channel and spatial reliability". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6309–6318.

[43] Liangchen Luo et al. "Adaptive gradient methods with dynamic bound of learning rate". In: *arXiv preprint arXiv:1902.09843* (2019).

[44] Y. Lv et al. "Traffic Flow Prediction With Big Data: A Deep Learning Approach". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873. ISSN: 1558-0016. DOI: 10.1109/TITS.2014.2345663.

[45] Alexander Neubeck and Luc Van Gool. "Efficient non-maximum suppression". In: *18th International Conference on Pattern Recognition (ICPR'06)*. Vol. 3. IEEE. 2006, pp. 850–855.

[46] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 427–436.

[47] Ki-Yeong Park and Sun-Young Hwang. "Robust Range Estimation with a Monocular Camera for Vision-Based Forward Collision Warning System". In: *TheScientificWorldJournal* 2014 (Jan. 2014), p. 923632. DOI: 10.1155/2014/923632.

[48] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). DOI: 10.1109/cvpr.2017.690. URL: http://dx.doi.org/10.1109/CVPR.2017.690.

[49] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[50] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[51] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[52] Hamid Rezatofighi et al. "Generalized Intersection over Union". In: (2019).

[53] Mohammad Amin Sadeghi and David Forsyth. "30hz object detection with dpm v5". In: *European Conference on Computer Vision*. Springer. 2014, pp. 65–79.

[54] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). DOI: 10.1109/cvpr.2018.00474. URL: http://dx.doi.org/10.1109/CVPR.2018.00474.

[55] Robert E. Schapire and Yoram Singer. "Improved Boosting Algorithms Using Confidence-rated Predictions". In: *Machine Learning* 37 (1999), pp. 297–336.

[56] Jürgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *CoRR* abs/1404.7828 (2014). arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

[57] Mohammad Javad Shafiee et al. "Fast YOLO: A fast you only look once system for real-time embedded object detection in video". In: *arXiv preprint arXiv:1709.05943* (2017).

[58] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *On a Formal Model of Safe and Scalable Self-driving Cars*. 2018. arXiv: 1708.06374 [cs.RO].

[59] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[60] A. Swief and M. El-Habrouk. "A survey of Automotive Driving Assistance Systems technologies". In: *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. 2018, pp. 1–12. DOI: 10.1109/IDAP.2018.8620826.

[61] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10781–10790.

[62] Robert L. Thorndike. "Who belongs in the family?" In: *Psychometrika* 18.4 (1953), pp. 267–276. ISSN: 1860-0980. DOI: 10.1007/BF02289263.

[63] Jasper RR Uijlings et al. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[64] Ren Wu et al. "Deep Image: Scaling up Image Recognition". In: *ArXiv* abs/1501.02876 (2015).

[65] Fisher Yu et al. “BDD100K: A diverse driving dataset for heterogeneous multitask learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2636–2645.

•