Object Detection with Two-stream Convolutional Networks and Scene Geometry Information

Binghao Wang

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master degree in
Electrical and Computer Engineering

uOttawa

School of Electrical Engineering and Computer Science
Falculty of Engineering, University of Ottawa
Ottawa, Ontario

Abstract

With the emergence of Convolutional Neural Network (CNN) models, precision of image classification tasks has been improved significantly over these years. Regional CNN (RCNN) model is proposed to solve object detection tasks with a combination of Region Proposal Network and CNN. This model improves the detection accuracy but suffer from slow inference speed because of its multi-stage structure. The Single Stage Detection (SSD) network is later proposed to further improve the object detection benchmark in terms of accuracy and speed. However, SSD model still suffers from high miss rate on small targets since datasets are usually dominated by medium and large sized objects, which don't share the same features with small ones.

On the other hand, geometric analysis on dataset images can provide additional information before model training. In this thesis, we propose several SSD-based models with adjusted parameters on feature extraction layers by using geometric analysis on KITTI and Caltech Pedestrian datasets. This analysis extends SSD's capability on small objects detection. To further improve detection accuracy, we propose a two-stream network, which uses one stream to detect medium to large objects, and another stream specifically for small objects. This two-stream model achieves competitive performance comparing to other algorithms on KITTI and Caltech Pedestrian benchmark. Those results are shown and analysed in this thesis as well.

Acknowledgement

I would like to thank my supervisor, Professor Robert Laganière for his guidance. I got so many valuable experience from getting involved in all those inspiring projects under his supervision during my master study. I was provided with many opportunities to work on different interesting tasks both singlehandedly or with other colleagues. Likewise, I would give my sincere gratitude to Dr. Yong Wang for his suggestions to my research. Finally, I need to express my profound gratitude to my parents and my girlfriend, Qianyao Yang, for their unfailing support and continuous encouragement throughout my years of study and the process of researching. This accomplishment would not have been possible without them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Object detection has been a hot topic among computer vision researchers over decades. As an important component for many applications such as self-driving vehicles, intelligent surveillance, robotics, it has attracted even more attention within the community and industry in recent years. The primary goal of object detection is to locate all primary objects (such as pedestrians, cars, etc.) on a single image. The locations of the objects will be indicated by the bounding boxes, as well as their classes and sizes. Many applications can be built with those information since they are crucial for understanding an image.

Many companies are devoting their resourses to researching on Autonomous Driving System, in which object detection is one of the core modules (Figure 1.1). For each self-driving system with visual sensors, it's a primary step to understand the surrounding scene and locate every objects ahead, such as cars, vans and pedestrians etc. The detection algorithm must exhibit high accuracy, efficiency and robustness. Given the object's position and size information generated from a detection algorithm, other modules can calculate the distance between the object and the sensor. It will help the system make further decisions.

As one of the major fields benefited by deep learning and neural network, object detection have also been in transition from using hand-crafted features extraction to deep network. These new techniques are achieving much higher accuracy comparing to the traditional methods. The development of general object detection models open the door to a large number of new applications.

Figure 1.1: Object (Pedestrian) detection demo.

1.1 Problem Statement

For every self-driving system with visual sensors, the first step is always understanding the scene and parsing the input image. The information extracted from the first step is crucial since very little disturbance can tremendously affect the following steps in the system. Nowadays, various models have been proposed to achieve better performance. Lots of companies are also developing their own detection algorithms to better fit their system. Remarkable progress has been made with the help of deep neural networks and other machine learning algorithms. However, it's still necessary and there's still plenty of room to improve both accuracy and efficiency of these approaches.

We aim to improve the performance of an end-to-end deep neural network detection model called SSD (Single Shot MultiBox Detection) [23] to generate bounding boxes of target objects from a single image with higher accuracy without impacting its efficiency.

1.2 Motivation of the Problem

We target Advanced Driver-Assistance Systems (ADAS) and self-driving vehicles so we decided to focus on object detection especially on vehicles and pedestrians.

Object detection used to use a combination of hand-crafted feature extraction and sliding window technique. Commonly used features like SIFT [36], SURF [2], HOG [7], etc. are extracted to characterized the local regions. In the past few years, the increase of GPU computational power has boosted the development of machine intelligence and deep learning, which are now widely researched and deployed in various industrial applications. Hand-crafted features are gradually replaced by deep neural networks as they can achieve much better performance on different tasks. Their development has benefited from the emergence of different large datasets like IMAGENET [27].

Various models were proposed based on deep Convolutional Neural Networks (CNNs). One of the most famous is Single Shot Detecter (SSD) [23]. Other than other multi-stage detection frameworks, SSD models elimimate the region proposal procedure (will be discussed in Section 2.2.2.1) and calculate everything in a single network. Those features make SSD outperform most recent models including R-CNN series models [16, 15, 26]. However, even with very deep base network, the SSD model still have difficulty detecting objects with a wide range of sizes. It's mainly due to the architecture of SSD which impose on small and large objects to share the same features. To overcome this problem, we try to build a deep network based on SSD but with a sub-network especially designed for small object detection purpose.

1.3 Contributions

We propose a deep two-stream network based on SSD for object detection. We also explored the geometric feature of object distribution and use that to increase our detection speed and reduce model size. We test our model performance on vehicle detection on KITTI dataset.

In general, the contributions of this thesis are:

- We propose a deep convolutional neural network with two streams for object detection. Our model outperformes the original SSD model especially on small object detection.
- Explain the reason of adding another stream in our model as well as its structure in detail.
- Use statistical analysis on training data, i.e. KITTI and Caltech Pedestrian in our case, which helps us reduce the model size and boost detection accuracy.

1.4 Thesis Outline

This thesis is organized as follows:

- Chapter 2 introduces the background and related works on object detection models. We present two main categories of models: R-CNN based and single stage based. Our proposed model is a single stage model but also inspired by those R-CNN model architectures. We also present some popular object detection datasets at the end of this chapter.
- Chapter 3 introduces the basic SSD and presents our proposed models, which can be considered an improved version of SSD.
- Chapter 4 shows the preprocessing procedure for the dataset and the training configurations. We also explain our experiments and discuss the results in this chapter.
- Chapter 5 summarizes the thesis and presents its limitations and potential future works.

Chapter 2

Related Works

In the chapter, we review the development of object detection in recent years and present some famous object detection models, including traditional approaches and Deep Neural Network related approaches.

We emphasize on explaining R-CNN and SSD frameworks since many state-of-the-art object detection models are built on top of these two. They're both based on deep convolutional networks but with different architectures. Not only the basic ideas and architectures, we also introduce the evolution and some derivatives of these two model families in subsections. Our model is based on the original SSD but also inspired from some R-CNN models and algorithms.

Finally, we introduce the datasets we choose for vehicle and pedestrian detection, as well as other popular object detection datasets with street scene.

2.1 Non Deep Neural Network (DNN) approaches for object detection

In the 90s, feature descriptors are the primary techniques used to solve many computer vision problems. Those descriptors are then combined with machine learning algorithms like Support Vector Machine and Nearest Neighbours as an integral approach to solve those

problems. Boosting or cascade techniques are also introduced to enhance the performance of the descriptors.

Histogram of Oriented Gradients The HOG [7] is one of the most famous and successful feature descriptor uses hand crafted features. Images are divided into small regions called cells, and the histrogram of gradient directions is calculated for each cell. A group of adjancent cells is call a block and the block histogram is represented by the normalized group of histograms. HOG detector uses the sliding window technique and extracted features are then sent to a linear SVM detector.

Boosted Cascade Machine learning approaches has been adopted for visual object detection for decades. Boosted Cascade [34], a combination of boosting algorithm and cascade method, was proposed to achieve 15x faster speed than any other algorithms that time. A specific boosting algorithm called Adaboost is used to boost the performance of a weak learner. In this case it's used to select a small set of features and train the classifiers. There are originally 180,000 potential features and one of them is selected in each iteration. Then a series of classifiers are introduced into their algorithm. A large number of negative examples are discarded by the first weak classifier and the following classifers only need to compute on the ones which passed from the previous stage(s). It needs more computation on further stages, but huge amount of potential examples are already eliminated in the early stages.

Later Deep Cascade [1] is proposed using the same methodology but replacing the hand-crafted features with deep neural networks.

ACF++ Boosted decision tree is also an effecient approach to deal with large datasets. ACF++ [24] uses a combination of boosting framework and data augmentation techniques which are commonly seen in pre-training steps of CNN models. Not like CNN methods, ACF++ doesn't need large amount of external data for model training. Soft cascade with decision trees are employed as the weak learners while augmentation procedures such as color alteration, flipping, translation, cropping, occlusion are also implemented to increase

dataset size. This model finally achieves an average miss rate of 18%, which improves the Aggregate Channel Feature (ACF) detector by around 10%.

2.2 Deep Neural Network (DNN) related approaches for object detection

2.2.1 Basic layers in Convolutional Neural Network

In this section, we present four basic components of a CNN: convolutional layer, pooling layer, ReLU layer and fully-connected layer. Those are the building blocks of every deep convolutional neural networks as well as the base network in our proposed model.

Convolutional layer A convolutional layer consists a set of kernels with the same size. Given a certain input, it operates a 2-dimensional convolution between the kernel and the input. The output convolution results is further passed to the next layer in the network. For example (Figure 2.1), given a 4×4 input matrix without padding, if we convolute a 3×3 kernel on it with stride of 1, a 2×2 matrix will be generated as the output. The size of kernels are mainly 5×5, 3×3 and 1×1 in our experiments .

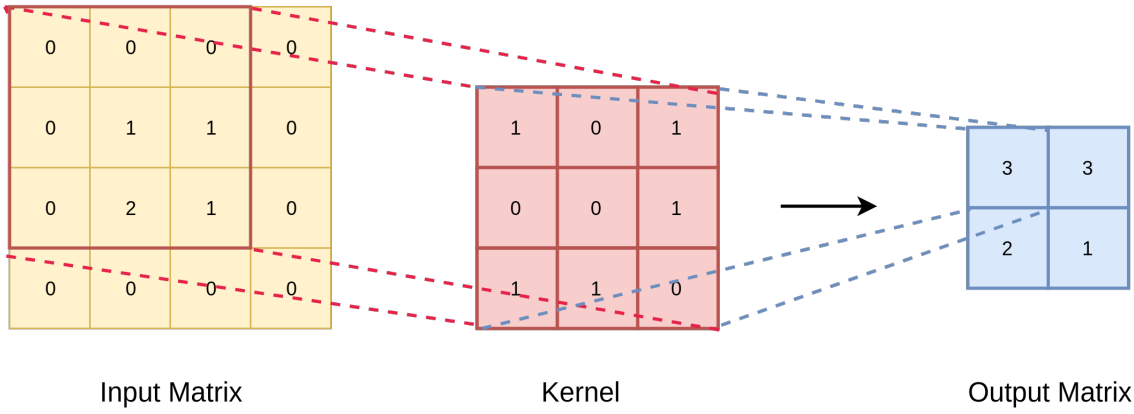

Figure 2.1: Convolutional layer illustration with input matrix of 4×4, kernel size of 3×3, stride of 1, an output matrix of 2×2 will be generated.

Pooling layer Different kinds of pooling layers are introduced into CNN's as down-sampling operations. The most widely used is max pooling and it simply pools the maximum value among a local region. If we conduct a 2×2 max pooling over a 4×4 input matrix with stride of 2, the output is then a 2×2 matrix as shown in Figure 2.2.

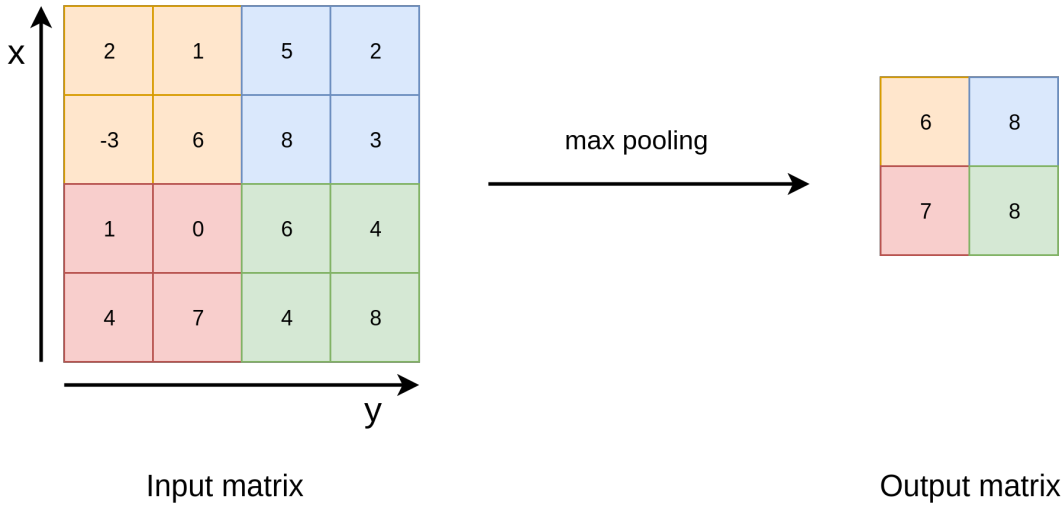

Figure 2.2: 2×2 max pooling layer.

ReLU layer Rectified Linear Units (Figure 2.3) apply elementwise activation function defined as

$$f(x) = \max(0, x) \tag{2.1}$$

Activation functions can introduce nonlinearity into the model, which enables deep networks to be adapted to more complicated tasks.

Other functions can also be used as activation function like hyperbolic tangent:

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.2}$$

and the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Comparing to those functions, ReLU converges faster as the slope doesn't plateau (saturate) as x goes large. ReLU is computed several times faster than the other activation functions [21] as well and is the most widely used.

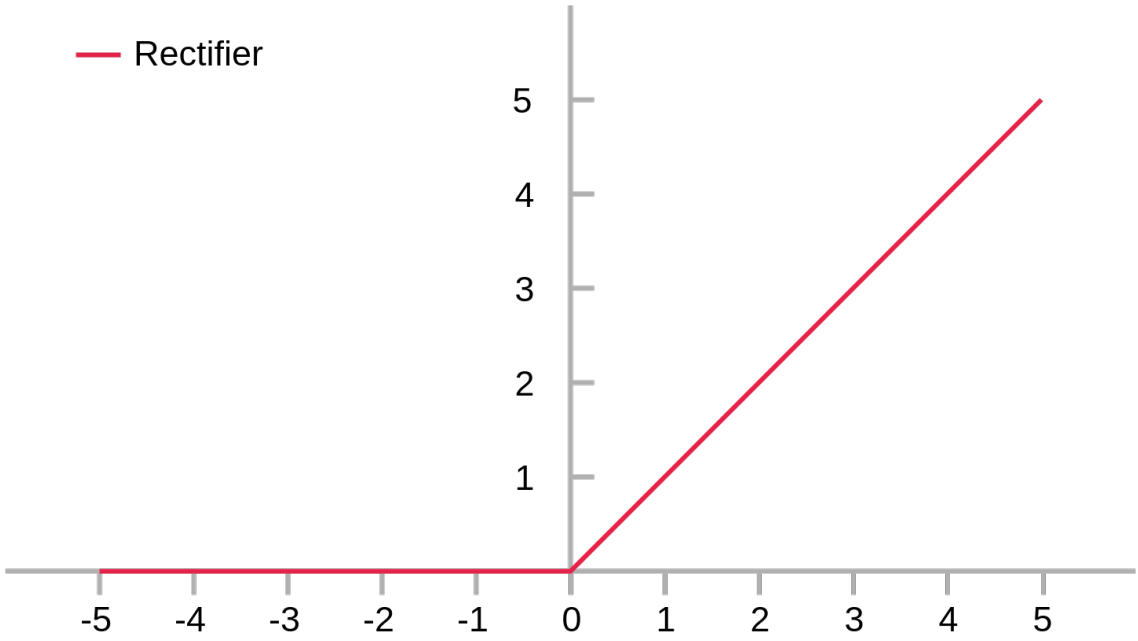

Figure 2.3: ReLU layer.

2.2.2 R-CNN series

In this section, we review the development of the region-based R-CNN series models and their derivatives which inspire the creation of our model. It has evolved since the first generation through different phases: R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN (Figure 2.4). In general, we

- present the evolution of this efficient model through different stages
- review some derivatives based on R-CNN models which can improve the performance

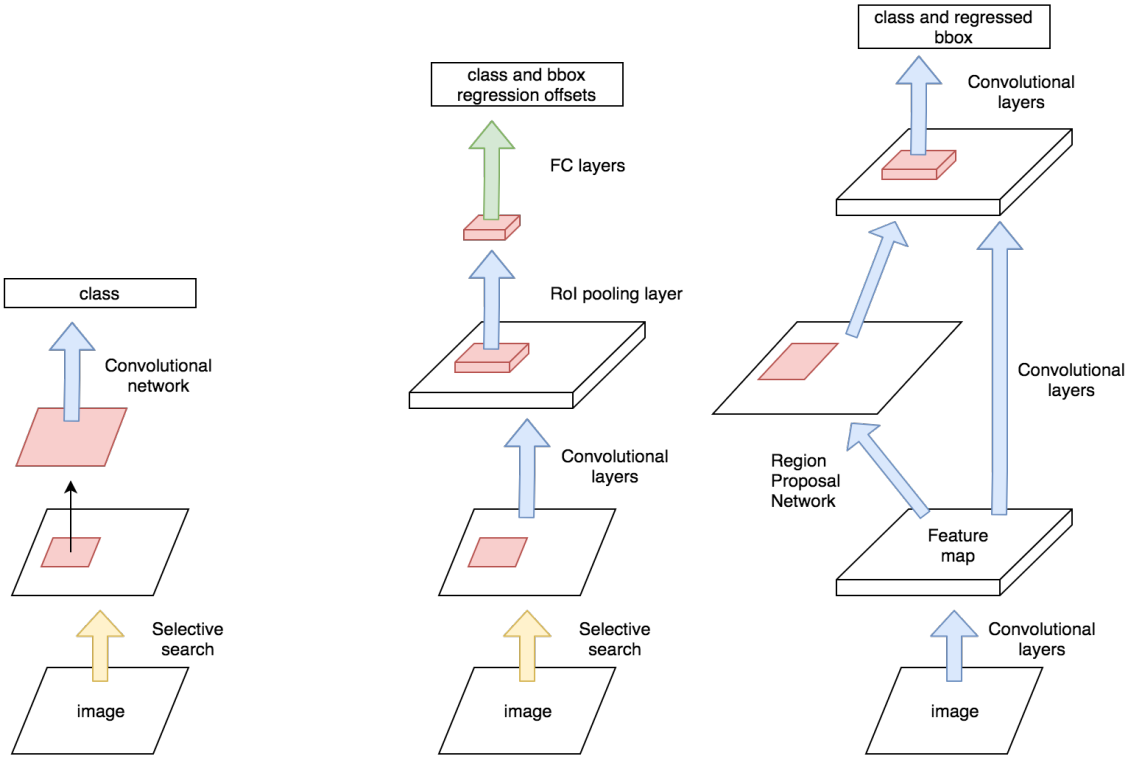

Figure 2.4: R-CNN [16] (left), Fast R-CNN [15] (center) and Faster R-CNN [26] (right) network architectures comparison. R-CNN is simply an image classifier on each proposals from seletive search. Fast R-CNN uses Regions of Interest (RoI) pooling to avoid repetitive inference computation. Faster R-CNN use Region Proposal Network (RPN) to propose region directly from feature map.

2.2.2.1 Fast, Faster and Mask R-CNN

R-CNN Since CNNs [27] achieved the best performance on image classification, people have been given thoughts on using them for object detection tasks. The earliest and most intuitive models are the R-CNN family. Given a single image, R-CNN propose nearly 2000 regions using Selective Search [33]. Those regions are then passed to a modified version of AlexNet, the winner of ImageNet 2012 competition for image classfication, to extract features. A Support Vector Machine (SVM) [6] is also attached at the end to classify each region proposal.

The very first R-CNN model is a combination of three modules: region proposal module by selective search, classification module by CNN and SVM and regression module to tighten the bounding boxes. The three-module architecture makes the pipeline very hard to train. It also requires a forward pass (around 2000 in total) to the deep CNN for each proposed region. Still, the original R-CNN model achieves much higher accuracy on object detection benchmarks comparing to its peers at that time. It achieves a mean Average Precision (mAP) of 53.7% on PASCAL VOC 2010 while [33] reports 35.1% mAP using spatial pyramid and bag-of-visual-words approach. It also achieves 31.4 mAP on ILSVRC2013 while OverFeat [28] had the previous best result at 24.3%.

Fast R-CNN Those problems mentioned above were soon realised by the R-CNN [16] creator. RoIPool (Region of Interest Pooling) is then proposed to solve the problem that huge CNN computations are wasted for overlapped regions proposals. In Fast R-CNN, an input image is passed to deep CNN once and those proposed regions are directly projected to the feature map extracted by CNN. At the end of network after the RoI pooling layer, original SVM is also replaced by a softmax activation function as the classifier and a bounding box regressor.

Fast R-CNN achieves higher accuracy with much faster speed comparing to R-CNN. It trains VGG16 network $9\times$ faster and is $213\times$ faster at test time. The training of Fast R-CNN is one-stage and all network layers can be updated during training. The disk space for caching features in R-CNN are no longer needed as well.

Faster R-CNN Even with more than 100 times speedup, Fast R-CNN [15] still waste too much time on the selective search. Faster R-CNN is then proposed to overcome the bottleneck of the overall process. Instead of using the computationally expensive selective search method, it use the extracted CNN feature maps for nearly cost-free region proposals.

An additional network named Region Proposal Network (RPN) is included to fulfill the region proposing task. It works by passing a sliding window over the CNN feature map and at each window, generating k potential bounding boxes and scores.

Faster R-CNN has near real-time detection rate and is now 5 fps on a GPU, while achieving state-of-the-art accurary on PASCAL VOC 2007, 2012 and MS COCO datasets.

Mask R-CNN Mask R-CNN [17] is the lastest model of the R-CNN series. It combines an additional branch to Faster R-CNN which outputs a binary mask indicating if a given pixel is part of an object. It focuses on image segmentation and is beyond the scope of this thesis. But it shows the R-CNNs are highly efficient models with extensible feature.

Mask R-CNN outperforms all existing algorithm on all three tasks (detection, key-point finding, segmentation) MS COCO dataset.

2.2.2.2 Other R-CNN model derivatives

The RPN is naturally a detector for single category detection. Based on this, Zhang et al. [38] discover that downstream classifier in Faster R-CNN degrades the results of pedestrian detection. They argue it may be caused by insufficient resolution of feature maps for small instance and no procedure for hard negative mining. They propose a model contains a combination of RPN (to generate candidate boxes and feature maps) and boosted forest (to classify the proposals using these features). They present competitive accuracy and speed on several benchmarks (Caltech[9], INRIA [8], ETH [11] and KITTI [14]).

In natural street scenes, objects like vehicles and pedestrians have a wide range of sizes, and usually the overwhelming majority instances in a dataset are small objects. However, detectors trained on those datasets will most likely perform well on large objects but significantly worse on small objects. [22] claims that instances with different spatial scales have dramatically different features. Detector performance can be severely hurt by the large variance in instance scales. Thus, they develop a Scale-Aware Fast R-CNN (SAF R-CNN, Figure 2.5) with two sub-networks, which is trained to extract large and small objects' feature separately. Their outputs are then combined via a gate function based on the size of the object. Two sub-networks also share the first few layers to alleviate the computational cost of the model. The architecture of SAF R-CNN is based on Fast R-CNN and it uses an ACF-detector to generate objects proposals.

Another way to improve the detector performance on small objects and hard positives is to use contextual information. A new approach named Contextual Multi-Scale Region-based CNN (CMS R-CNN) is proposed to detect objects with heavy occlusion, pose variations and low resolution, etc. It outperforms other models on WIDER FACE dataset [37], which contains high variability of objects' conditions. The model is based on VGG-16 [30]

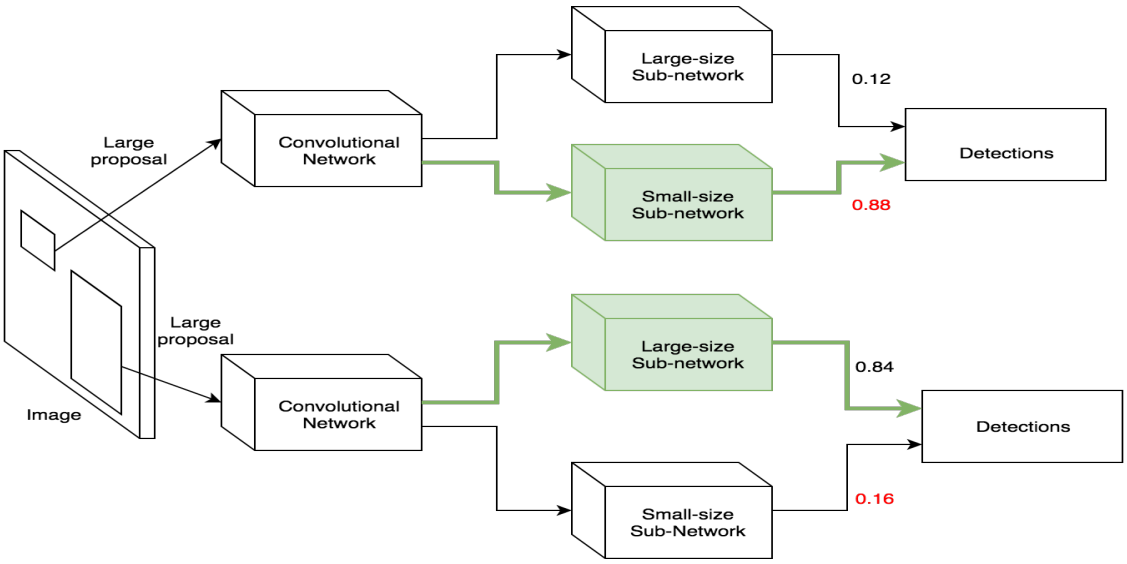

Figure 2.5: SAR R-CNN network architecture [22]: two sub-networks are trained to detect large-size and small-size objects separately. The overall detection result is a weighted combination of outputs from two sub-networks. The weight depends on the size of input proposal. For example, if the proposal is small, small-size network has higher weight.

and it consists of a Multi-Scale Proposal Network (MS-RPN) and a Contextual Multi-Scale Convolutional Neural Network (CMS-CNN). In the CMS-CNN, the face features extracted from earlier stage of the network and the body features from later stage are processed in parallel and combined for the final outputs. Their solution of combining global and local features is inspired by human visual system and is good at detecting tiny heavily occluded and low-resolution objects.

2.2.2.3 Performance improvement on R-CNN with special training strategy

Instead of model structure innovation, Sun et al. [31] use a set of training strategy to achieve better results. With a VGG16 pretrained on ImageNet as the basic network, they train a Faster R-CNN model on WIDER FACE dataset. Hard negative mining is performed as the second step of their training procedure. Next they finetune the model on FDDB dataset and combine the features from Conv3_3 (3rd convolutional layer in stage 3, ResNet acronym, see Section 3.1.2), Conv4_3 (3rd convolutional layer in stage 4) and Conv5_3

(3rd convolutional layer in stage 5) layers to feed into RoI pooling. This feature concatenation scheme is similar with [30] but simpler.

There is also other researches focusing on the dataset itself. The real world scenarios are much more complicated and there are only limited training samples in datasets. Since the more training samples we have the better performance deep networks can achieve, there's always a motivation to have a larger training dataset. It's very common to enlarge the dataset using label-preserving transformations [29][3][4], other data augmentation methods, such as horizontal reflections and RGB channel intensities altering, are also applied in [21]. The Adversarial Fast-RCNN [35] is then proposed to generate examples with occlusions and deformations that are very difficult for the object detector to classify. Their work makes a performance boost of 2.3% mAP on VOC07 and 2.6% on VOC2012 compared to the Fast-RCNN pipeline.

2.2.3 Single Stage Detector

An R-CNN model is really a combination Selective Search, CNN, SVM and a regression network. The large detection pipeline achieves high accuracy but it is very computational expensive. Even with its evolved versions (Fast and Faster R-CNN), inference speed is still keeping them far away from real-time applications.

With this problem in mind, several single stage detection models are proposed to have real-time efficiency. Some of them can meanwhile achieve competitive or even better performance comparing to large R-CNN models. In this section, we present some representative single stage models, including the one (SSD) we use as our base network. In general, we

- present some single stage models and compare their differences with R-CNN based models
- review some derivatives which further improves the model performance

Overfeat A new model with a special feature extractor named Overfeat [28] is proposed to do localization and classification simultaneously. The classification part is still a very deep convolutional network, but the localization is done by training a regression layer on

top of feature layers to predict the locations. To increase inference speed, they run the classifier and regressor across all locations and scales at the same time. Sliding window approach is used to generate confidence values for each window and those windows are regressed to the most confident area.

Layer 1–5 are used for feature extraction, layer 6–8 is for classification. The whole image is directly passed to the filter instead of the proposed regions. This is also far more efficient than a sliding window, which slides a fixed size filter over the image and add all the results. For the classfication filters, a fixed-size 5×5 input is always provided with single pixel shifts (Δ_x, Δ_y) on feature map 5 to keep the object location information.

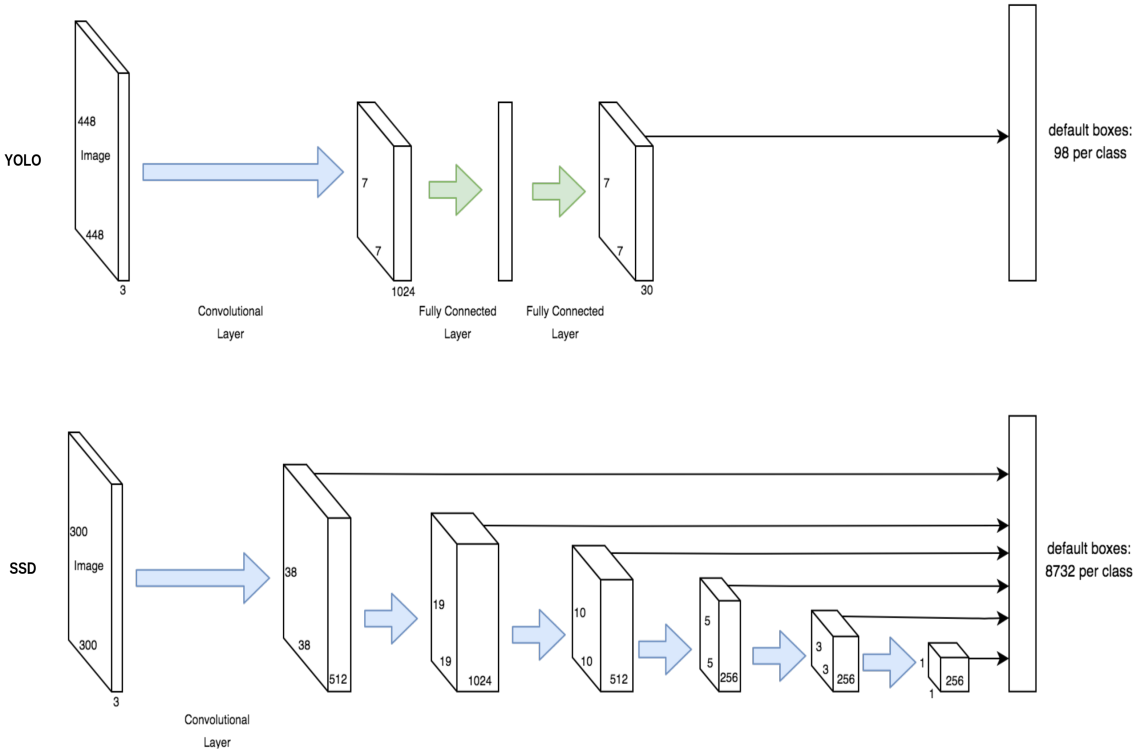

Figure 2.6: YOLO [25] & SSD [23] network architectures comparison: SSD extract features from several layers at the end while YOLO only extract features from the one layer. SSD also generates much more defaults boxes than YOLO, while still having higher inference speed.

YOLO (You Only Look Once) As implied by its name, YOLO [25] is extremly fast, it can process images at 45 frames per second. A smaller version, Fast YOLO, can even process 155 frames per second. To accomplish this real-time efficient, YOLO uses a single CNN network for both classification and objects locating. The network architecture is based on GoogLeNet model [32] for image classification. It consists of 24 convolutional layers and 2 fully connected layers. The model convolutional layers are pretrained on ImageNet with resolution of 224×224 and then trained on the PASCAL VOC dataset. The final detection model takes input image of 448×448 and outputs a $7 \times 7 \times 30$ tensor. The model divides the image into a 7×7 grid and each grid predicts 2 bounding boxes, as well as 20 class probabilities. A bounding box contains 5 predictions of object coordinates (x, y) and size (w, h) as well as its confidence. Since there are 7×7 grid cells and each predicts 2 bounding boxes, we end up with 98 bounding boxes with 20 class probabilities in total. Most of them have very low confidence scores and NMS (Non-Maximum Suppression) is applied to suppress multiple bounding boxes on the same object.

YOLO is a simple and straightforward approach as it both predicts bounding boxes and class probabilities on the final layer. This network architecture provides it real-time processing ability. However, since each grid cell from last layer only predicts 2 bounding boxes, it's very difficult for YOLO to detect small and grouped objects. Objects with unusual size or aspect ratio is also hard to detect.

SSD (Single Shot multibox Detection) Another one-stage model that eliminates region proposal generation is SSD [23]. It has similar architecture as YOLO but with faster inference speed, while achieves high accuracy as the slow techniques like Faster R-CNN. Instead of predicting bounding boxes and class probabilities from the last feature layer in YOLO, it has several feature layers at the end of the base network. Each feature layer in SSD has a fixed set of cells, each cell also has a fixed set of bounding boxes in different sizes and aspect ratios. Class prediction scores and offsets of default boxes are also produced from SSD to generate the final detections. Details of SSD architecture and training procedure are explained in sub-section 3.1.

SSD model derivatives Kim et al. [20] compare the performance of YOLO and SSD in their recent work, as well as their customized SSD model. They use VGGNet16 as the base network pre-trained on ILSVRC CLS-LOC dataset and fineture it on KITTI dataset. It shows that SSD achieves higher accuracy and recall rate than YOLO. As images in KITTI dataset are about 370×1250 but their model takes 300×300 inputs, the objects in training patches, which is randomly sampled by SSD, can be long and narrow like Figure 2.7. They add extra aspect ratios for default boxes to solve this problem. Additional pedestrian samples from other datasets are also added into the training set.

Figure 2.7: Example of SSD training sample distortion in KITTI.

In [10], Du et al. proposed deep neural network fusion approach (Figure 2.8) to improve detection performance, especially small-size and occluded objects. They first use an SSD network as their object candidate generator by lowing its detection confidence threshold to 0.01, which will generate massive amount of candidates. All candidates are passed to the classification networks in parallel along with their confidence scores and labels. The original score is boosted if the classfier outputs high confidence score. Otherwise, the score will be decreased by a scaling factor.

DSSD (Deconvolutional SSD) With this single feed-forward convolutional network, the SSD detection framework achieves huge improvement on speed and accuracy. To further improve the detection accuracy, Fu et al. [13] proposed the DSSD (Deconvolutional Single

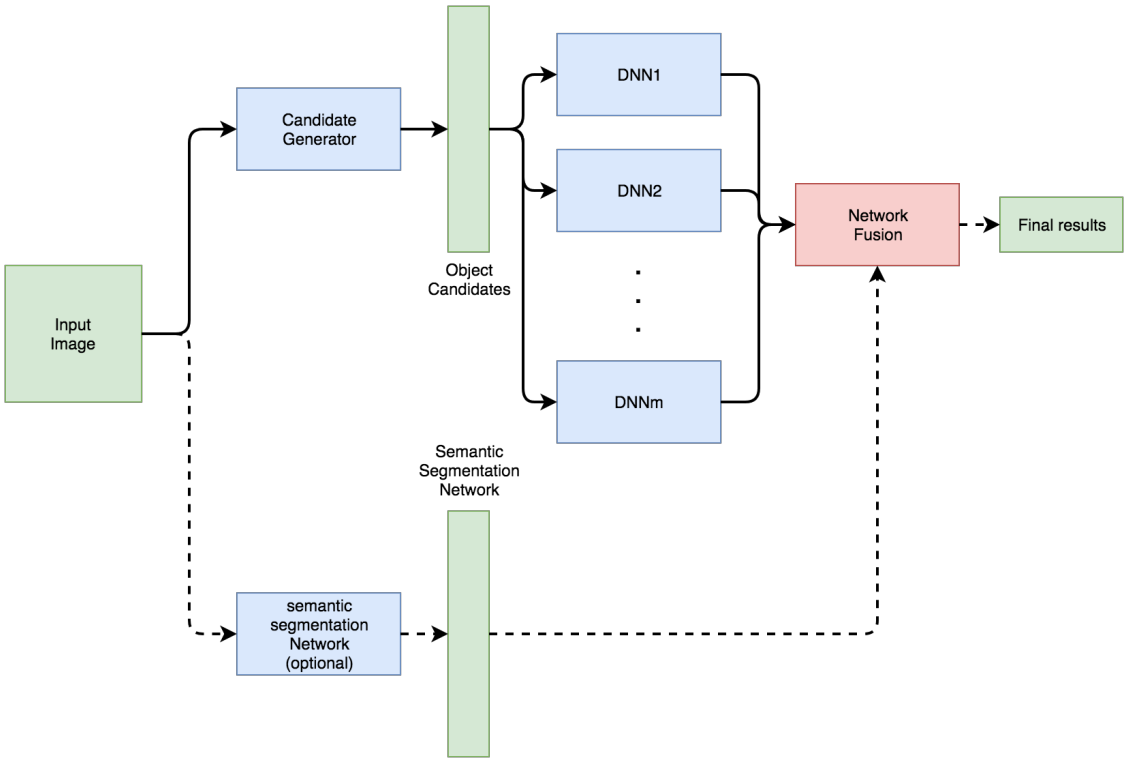

Figure 2.8: Fussed DNN [10] pipeline

Shot Detector) to project large scale context on small object feature, which extracted from additional deconvolutional layers at the end of an SSD network. Deconvolutional modules are introduced in the network to combine largescale and smallscale features. They use Residual-101 [18] instead of VGG as their base network, which provides better features. It's still a one-stage network but it trades off some speed for higher detection rate, especially of small objects.

2.3 Datasets

As the emergency of supervised learning algorithms, datasets play an important role in model training and validation. For object detection tasks, datasets provide thousands of images as well as their annotations at least. Redundant information is also provided sometimes such as image sources, camera configuration, etc. With the publication of several

large datasets, researches can easily compare their algorithm performance with others. We introduce several datasets that has been widely used for object detection.

ImageNet Since 2010, an annual contest named ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is held for each researchers all over the world to compete with each other. They need to evaluate their algorithms on the given dataset. In ILSVRC 2012, there are already 10,000,000 hand-labeled images in more than 10,000 object classes in the training dataset. It was originally famous as an image classification and a new object detection task appeared since 2013. It is similar with PASCAL VOC Challenge [12] but has 200 basic-level categories.

KITTI KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) [14] is built to provide a benchmark that is familiar with real-world scenarios. It's widely used in autonomous driving algorithm and system researchers and developers. It contains annotated video clips captured in rural area and highway in a mid-size city. KITTI dataset has nine categories of objects including Car, Van, Truck, Pedestrian, Person_sitting, Cyclist, Tram, Misc and Don'tCare. Object truncation and occlusion is also annotated as well as camera calibration information. As in Figure 2.9, bounding boxes are shown with different colours, which indicating the different occlusion level.

Caltech Pedestrian Caltech Pedestrian (Figure 2.10) is a well-known dataset which consists of 10-hour video from a driving vehicle in an urban environment with pedestrian annotations. The annotation includes occlusion label and temporal correspondence between bounding boxes. We extract each frame from those videos in all 11 sets and follow the suggestion to split them in two parts: training (set00, set01, set02, set03, set04, set05) and testing (set06, set07, set08, set09, set10). As a result, there are in total $249,950$ images in the dataset, including $132,080$ images containing person objects.

CityScapes CityScapes [5] is also captured in real-world scenarios. The video sequences are recorded in streets from 50 different cities, which generates 5,000 images with high-quality annotations and 20,000 images with coarse annotations. CityScapes dataset is not

Figure 2.9: Examples in KITTI dataset.

only larger than previous datasets in size, but it also has richer annotations and more complex scenes.

Figure 2.10: Examples of Caltech Pedestrian.

Chapter 3

SSD Architecture and Scene Geometry Analysis

In this chapter, we present the original SSD network architecture in detail, i.e., the base network and multibox detection module. Then we present our statistical analysis results on KITTI and Caltech Pedestrian dataset. Finally we propose our customized network as well as the two-stream network.

3.1 SSD architecture

We present the basic SSD architecture in this sub-section. We also give details of parameter setting and training procedure used in original SSD model [23].

3.1.1 SSD architecture

As is shown in Figure 3.1, SSD is a single stage network with multi-scale feature extraction layers. Class scores and offsets are predicted for each cell position on the feature maps in the multibox detection module at the end of network.

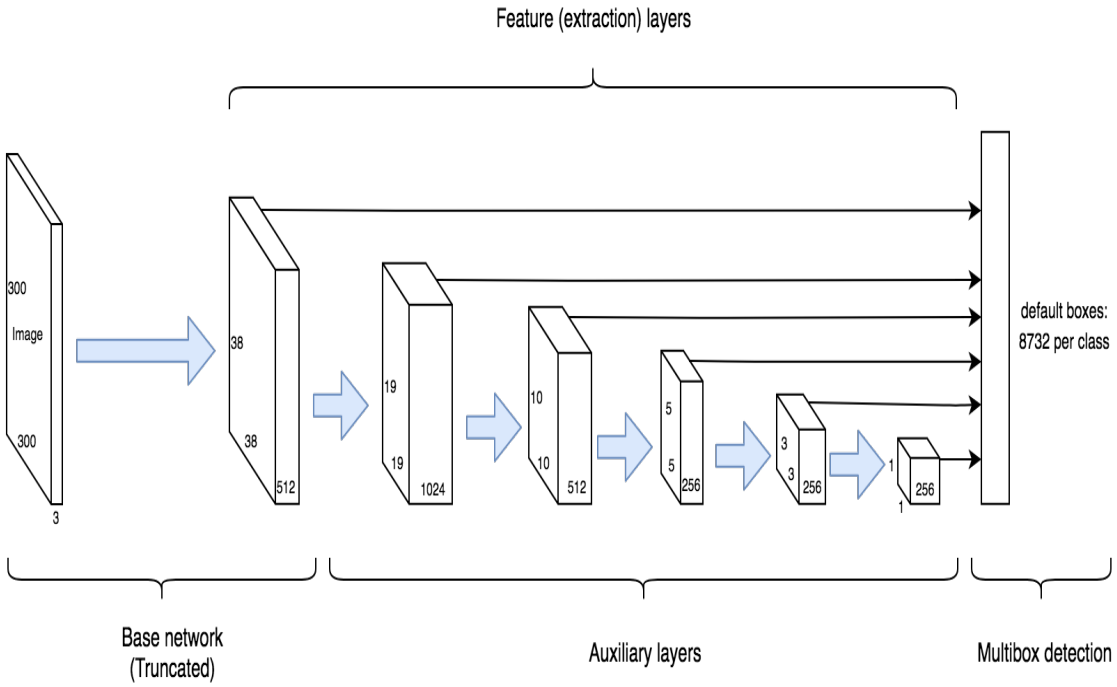

Figure 3.1: SSD [23] Architecture.

3.1.2 Base networks

As an important componenet of SSD architecture, base networks are considered as feature extractors, which provide high-level features for the following layers. It is crucial to select a suitable network as it will significantly affect the performance of the whole architecture. These networks are mostly designed for classification tasks and usually the deeper the network is, the better features are extracted from original images. However, the inference time and model size will most likely increase as the tradeoff. In this sub-section, we review some of most popular convolutional networks which can be used as feature extractors in SSD architecture.

VGG VGG [30] network (Figure 3.2) is proposed by Visual Geometry Group from University of Oxford in 2014 for large scale image recognition. It is used as the base network

at the first place when the SSD model was proposed. This network is the winner of ImageNet Chanllenge 2014 (first in localisation and second in classification). A fixed-size RGB image of 224×224 is required as the input of the network. One basic stage contains a stack of convolutional layers followed by a max-pooling layer. The convolutional layers have relatively small receptive field (3×3) while the max-pooling layers select the largest value in a 2×2 window with a stride of 2. After five stages of convolutional layers, 3 fully connected layers are attached. The first two have 4096 channels and the last one has 1000 channels for 1000 classes of ILSVRC dataset. Finally a soft-max layer is at the end of the network.

The outstanding performance of VGG series shows the importance of deep network. Other models based on VGG are also benefited by this very deep architecture.

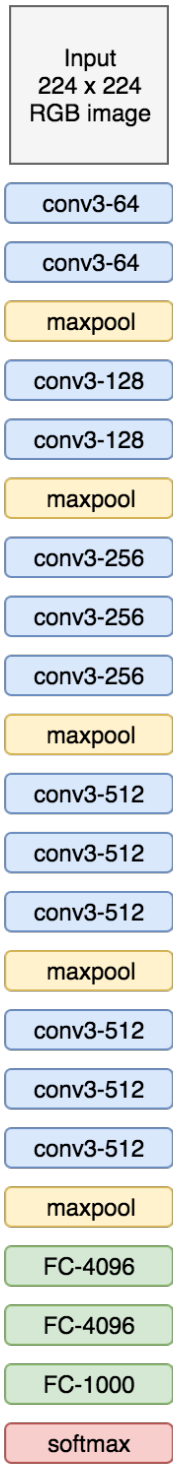

Figure 3.2: VGG16 network [30]. The convolutional layer configuration is denoted as "conv(kernel size)-(kernel number)". For example, conv3-64 means that layer has 64 kernels with the size of 3×3.

Residual Network (ResNet) Better performance are achieved by the development of deeper networks. However, these deep networks are also very difficult to train due to the infamous problem of "vanishing gradients". It means the gradient norms become exponentially smaller as layers go deeper in the network. And small gradients cause parameters changing slowly. Then it takes plenty of time to converge.

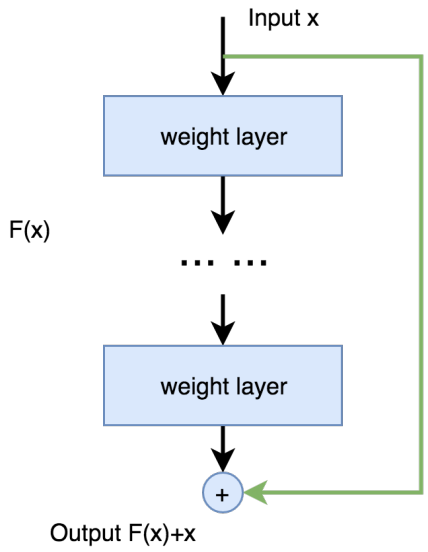

Figure 3.3: Residual learing block is a basic module in ResNet. The shortcut connection (in green) is used to bypass a stack of layers in order to overcome the "vanishing gradient" problem in deep networks. (Image from [18])

Fortunately, He et al. [18] present a deep residual learning framework to ease this problem. Shortcut connections are introduced to this framework to bypass a "stage" (a stack of convolutional, activation and normalization layers) as shown in Figure 3.3. The input block not only go through a stage and generate an output but also is passed directly to concatenate with the previous output.

The shortcut connections do not introduce any extra parameters or computational cost while it can ease the difficulty of training deep network. That is the main reason we adapt ResNet with 50 layers (ResNet50) to fit into our SSD architecture as the base network.

MobileNet With deeper networks well trained on a large dataset, better accuracy can be achieved but with speed compromised. A more effecient model called MobileNets is proposed in [19] to provide light weight deep networks especially for mobile applications.

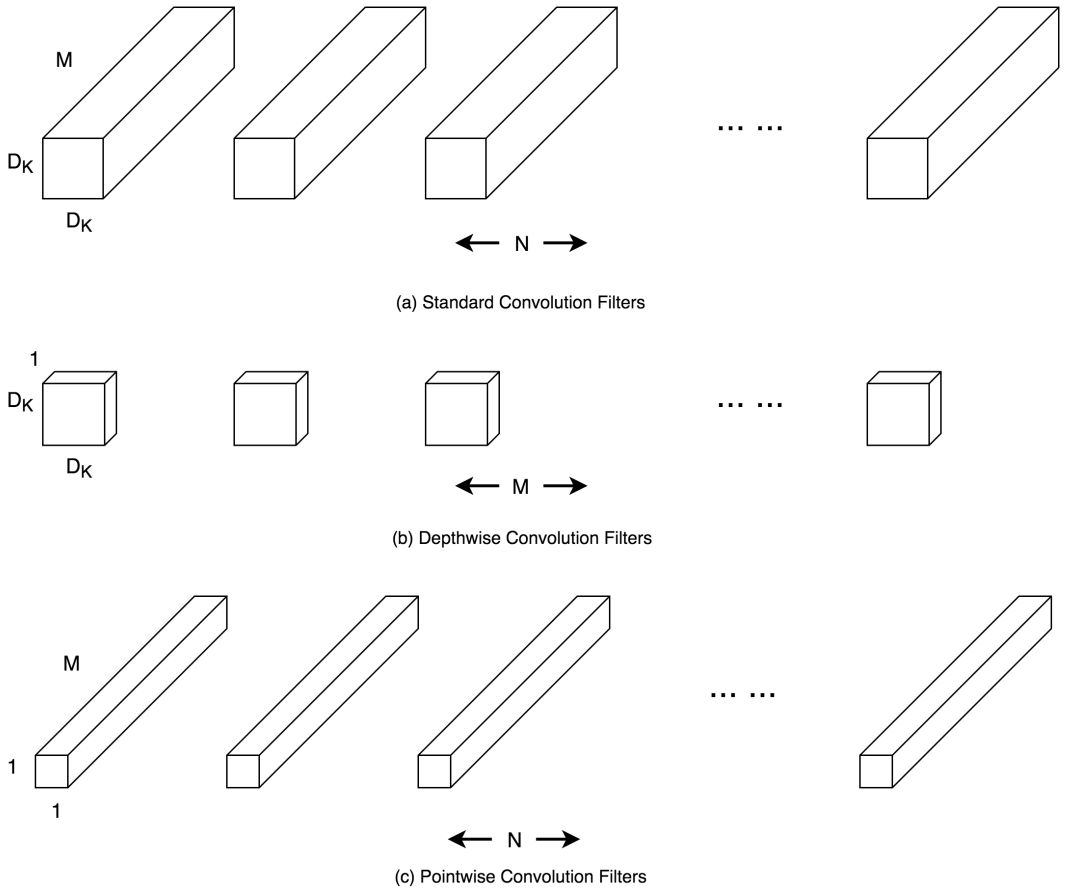

Figure 3.4: MobileNet kernels comparison. (a) standard convolution kernels, (b) depthwise convolution kernels and (c) pointwise convolution kernels. (Image from [19]).

The main innovation of that paper is Depthwise Separable Convolution. Depthwise convolution is very efficient comparing to standard ones. However, it can't create new features as it only filter through input channels. It leaves the combination task to the pointwise convolution at later stage.

A standard convolutional layer takes input of $D_F \times D_F \times M$ and outputs a $D_G \times D_G \times N$ feature map where D_F is the width of input, D_G is the size of width of output feature map, M is the number of input channels and N is the number of output channels. As in Figure 3.4, for a standard convolutional layer, the kernel size is $D_K \times D_K \times M \times N$ where D_K is the kernel width, M is the number of input channels and N is the number of kernels (output channels). Thus standard convolution with stride one has computational cost of

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F.$$

With depthwise convolution, the computational cost is $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$ and with pointwise convolution, is $M \cdot N \cdot D_F \cdot D_F$. The overall cost is

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F.$$

for the depthwise separable convolution.

We can find the reduction of the two step convolution is

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}.$$

If 3×3 kernels are used in the networks, we can save 9 time less computation with depthwise separable convolutions than standard ones.

Overall, we choose ResNet50 as our base network as it can achieve high detection accuracy while maintaining high inference speed as it's not too deep. And the structure of ResNet50 is not very sophisticated so that we can easily perform our model renovation and tests.

3.1.3 Multibox detection

For each cell on the feature maps of 6 different scales, default boxes with different sizes and aspect ratios are located at fixed positions. Class scores and offset predictions are given

for each default boxes. For each cell position on a $w \times h$ feature map, there are m different sizes and n different aspect ratios. Therefore, the number of default boxes generated on that feature map is $(m+n-1)wh$. For example (Figure 3.5), given a 4×4 feature map, if there are 2 sizes and 3 aspect ratios, the number of default boxes generated on this feature map is $(2+3-1) \times 4 \times 4 = 64$. At each default box position, class scores are calculated for all categories (background and vehicle for our binary classifer) as well as the deformations and shifts of the default boxes. It will be retained if a default box with a class score higher than the predefined threshold. While other boxes will be neglected. All selected boxes will be sorted by their scores and NMS (Non-Maximum Suppression) can be applied to suppress redundant detections.

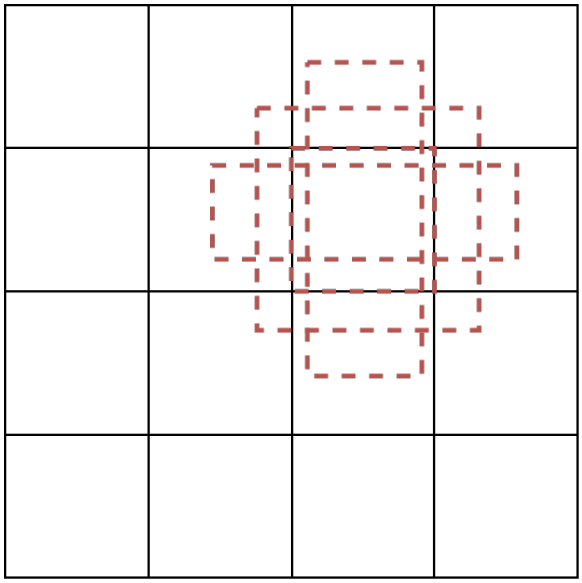

Figure 3.5: Multibox generation example. Given a 4×4 feature map, if there are 2 different sizes and 3 different aspect ratios configured, the number of default boxes generated on this feature map is $(2+3-1) \times 4 \times 4 = 64$.

Take SSD with VGG16 as example (Figure 2.6), the original base network is truncated and auxiliary layers are attached at the end. Details of multibox layers configuration can be found at Table 3.1. Given the scale range (minimal and maximal scale), multibox sizes are calculated using:

```
min_ratio = min_scale * 100
```

```
max_ratio = max_scale * 100
step = floor((max_ratio - min_ratio) / (num_layers - 2))
min_sizes = []
max_sizes = []
for (ratio = min_ratio; ratio <= max_ratio + 1; ratio += step):
        min_sizes.append(ratio / 100)
        max_sizes.append((ratio + step) / 100)
min_sizes = [100*min_scale / 2 / 100] + min_sizes
max_sizes = [min_scale] + max_sizes
```

This pseudocode is derived from the original SSD Caffe implementation and the default value for minimal and maximal scale is 0.2 and 0.9 respectively. Therefore, the multibox sizes 6 feature extraction layers are [.1, .14], [.2, .27], [.37, .45], [.54, .62], [.71, .79], [.88, .96], where the min_sizes are [.1, .2, .37, .54, .71, .88] and max_sizes are [.14, .27, .45, .62, .79, .96]. On a certain feature map, given multibox sizes [s_1, s_2] and ratios [r_1, r_2, r_3, (r_4, r_5)], the (width, height) sets of the multiboxes on one cell is calculated as ($s_1 r_1$, $s_1 r_1$), ($s_1\sqrt{r_2}, s_1\frac{1}{\sqrt{r_2}}$), ($s_1\sqrt{r_3}, s_1\frac{1}{\sqrt{r_3}}$), ($s_2 r_1$, $s_2 r_1$). For example, as $s_1 = 0.1, s_2 = 0.14, r_1 = 1, r_2 = 2, r_3 = \frac{1}{2}$ for the first layer, sizes of the multiboxes on the first layer can be (0.1×1, 0.1×1), ($0.1 \times \sqrt{2}, 0.1 \times \frac{1}{\sqrt{2}}$), ($0.1 \times \frac{1}{\sqrt{2}}, 0.1 \times \sqrt{2}$), ($0.14 \times 1$, 0.14×1).

Table 3.1: Multibox layers configuration for SSD with VGG16 network with input size of 300. In total there are $38 \times 38 \times 4 + 19 \times 19 \times 6 + 10 \times 10 \times 6 + 5 \times 5 \times 6 + 3 \times 3 \times 4 + 1 \times 1 \times 4 = 8732$ multiboxes.

layer index	feature map size	multibox sizes	multibox ratios	number of multiboxes
1	38×38	[.1, .14]	[1, 2, 1/2]	5776
2	19×19	[.2, .27]	[1, 2, 1/2, 3, 1/3]	2166
3	10×10	[.37, .45]	[1, 2, 1/2, 3, 1/3]	600
4	5×5	[.54, .62]	[1, 2, 1/2, 3, 1/3]	150
5	3×3	[.71, .79]	[1, 2, 1/2]	36
6	1×1	[.88, .96]	[1, 2, 1/2]	4

3.2 Scene Geometry Analysis

Size of objects varies in a given dataset. Objects closer to the camera usually seem bigger on the image than others far away from it. SSD can detect objects of different sizes based on the default boxes, which are generated on the feature extraction layers. Those default box sizes and aspect ratios are pre-defined for each layer. By performing scene geometry analysis, we can get the distribution of objects with different sizes and adjust those default boxes parameters to better fit the dataset.

In this section, we present the methods for statistical analysis on dataset as well as its results.

3.2.1 Statistical analysis of dataset

SSD model can be adapted to many object detection applications. As one of the most popular topics, vehicle detection is also one of the scenarios that SSD model can be applied to. We focus on vehicle detection in KITTI dataset in this section. KITTI has over 7,000 training images with annotations of nine classes (car, van, truck, pedestrian, person_sitting, cyclist, tram, miscellaneous, do-not-care) as well as the designated test set. Car, van and truck are merged as a single class “vehicle” and all others as background. Implementation can be adapted to multi-class detection easily. We aim to increase SSD model accuracy on KITTI by dataset analysis and model architecture renovation. KITTI includes images captured with a camera on a driving car. As we can imagine, objects near vanishing point on the image look smaller than the others near the edges (Figure 3.6).

Figure 3.6: Objects look smaller near vanishing point.

SSD can detect multi-scale objects at the same time with multiple feature extraction layers. Normally, there are 6 such layers at the very end of SSD model. Shallower layers generate feature maps for detecting smaller objects while deeper layers for larger objects.

There is a wide range of object sizes in KITTI. Other than the overall accuracy of SSD model, we also explore the performance of each feature layer separately.

With ResNet-50, first feature extraction is taken from the 6th convolutional layer on the 3rd stage and the second is from the 3rd convolutional layer on the 4th stage. Following layers in original ResNet-50 is removed and 4 new feature blocks are attached to the truncated network. A new feature block usually consists of a 1×1 convolutional layer with a stride of 1 and another 3×3 convolutional layer with a stride of 2.

As one of the pre-processing step, all images are resized to 1200×350 before feeding to the network. Given an input image of 1200×350 and forwarded to the feature extraction layers, 6 extracted feature map sizes are shown in Table 3.2 (can also see Figure 3.7).

Table 3.2: Sizes of 6 feature maps of SSD given input size of 1200×350.

Feature layer	Feature map size
1	75×22
2	75×22
3	38×11
4	19×6
5	10×3
6	5×2

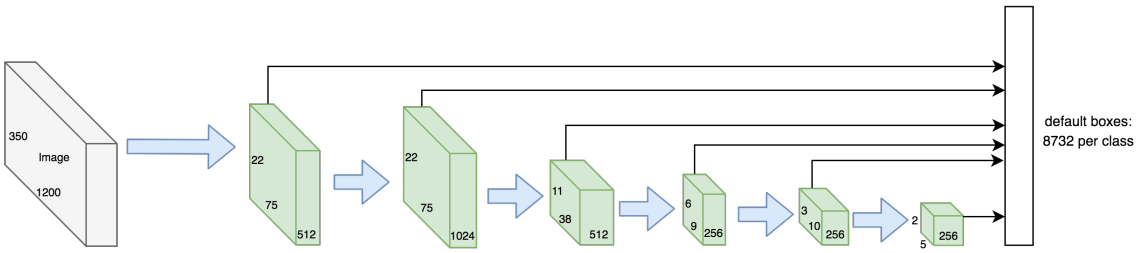

Figure 3.7: Given an input of 1200×350, 6 feature map sizes are 75×22, 75×22, 38×11, 9×6, 10×3, 5×2 respectively.

At every feature map position on each layer, default box sizes become larger as the layer goes deeper. As shown in Figure 3.8 and 3.9, we can see the default boxes projected to input image have different scales from all 6 feature maps.

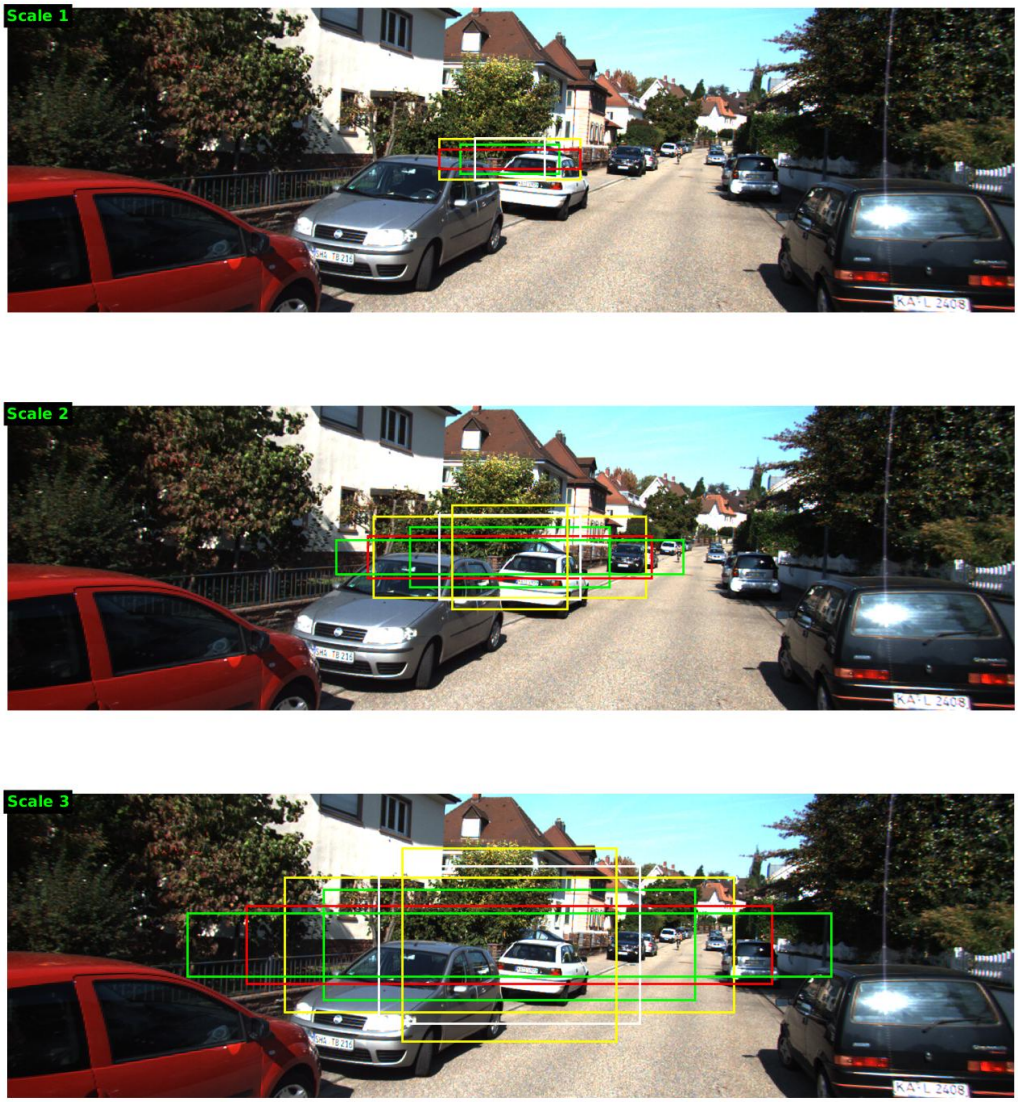

Figure 3.8: A set of default boxes generated from 1st to 3rd feature extraction layer (top to bottom). To be more intuitive, we mapped those boxes to original input image. In this example, with input image of 1200×350, default boxes on 1st layer are not small enough to detect the smallest objects.

Figure 3.9: A set of default boxes generated from 4th to 6th feature extraction layer (top to bottom). The default boxws are unnecessarily large on 5th and 6th layer.

To further investigate on the efficiency of feature maps, we keep record of the activated anchors on each of them. If a default box at certain position achieves score higher than the predefined threshold (0.5 in our case), then we call it activated. Given an input image,

there should be multiple activated default boxes around the objects. These are not the final bounding boxes since their size and location are not calibrated yet. But it's already precise enough to generate a general distribution of object proposals after each feature layer.

With that in mind, we traverse all images in the training set and accumulate the activated position on each feature map. The results from Figure 3.10 show that the small object proposals are gathered at the center and the large proposals are distributed at the lower half of the image as we expect. And there is no default box activated on feature layer 5 or 6.

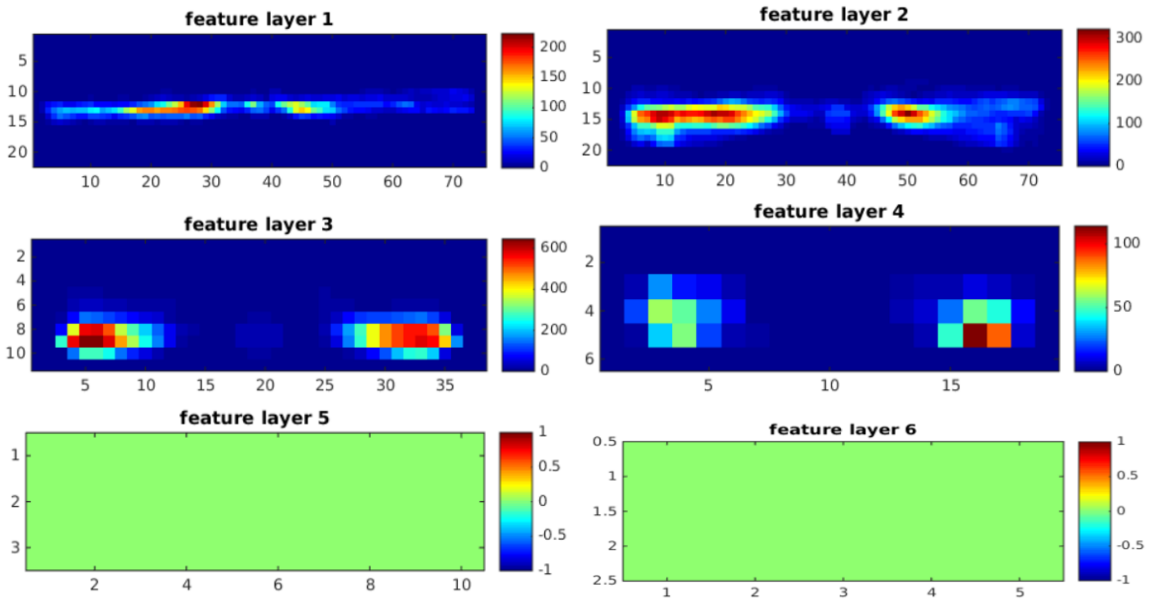

Figure 3.10: Activated default box distribution on each feature map.

3.2.2 Customized feature extraction layers

From our previous analysis, the last two layers can be removed as they don't contribute any object proposals on KITTI. Previous observation from Figure 3.9 is also indicates that bounding boxes of scale 5 & 6 are too large for our target objects.

We also test the inference performance with different amount of feature layers. The model is trained with all 6 feature layers but only part of them are used during inference period (Table 3.3). The result shows the first two layers play an significant role in this detection task. The feature layer 3, 4, 5, 6 seem not so important to increase mAP. In fact,

the mAP even drops a little bit after introducing the third and following layers. The reason is many false positives are also introduced by those layers while more proposals generated.

Table 3.3: Model mAPs with different number of feature layers.

Feature layer(s)	mAP
1	.65
1,2	.80
1,2,3	.79
1,2,3,4	.77
1,2,3,4,5	.77
1,2,3,4,5,6	.78

Therefore, we can remove the last two feature layers from original SSD model without any loss of accuracy. The parameters of default box sizes are also adjusted to a narrower range (see Table) from [.1, .961] to [.03, .5916] as we changed the minimal scale of first layer from .1 to .03. We keep the aspect ratios from original SSD in our experiment. This setting allows the model to generate smaller default boxes at the first feature layer, as well as to eliminate the redundant computations. Given a 1200×350 input image, default boxes generated from our customized model are much smaller thus they are more likely to match those small objects (Figure 3.11).

Figure 3.11: Visualization of customized network default box sizes at each cell on feature maps given input size of 1200×350.

This customized SSD model achieves higher detection rate on small objects with those size-adjusted default boxes of first feature layer. To visualize, distributions of objects with different heights are plotted (see Figure 3.12 and 3.13). More proposals are generated from each layer comparing to original ones. All customized layers now tend to detect smaller objects. We can see that there is no object proposals with a height less than 30 px in original model. However, around 200 of those proposals are made on the first layer in our customized model. The second layer of our customized model now covers all the detections from the first layer in original model.

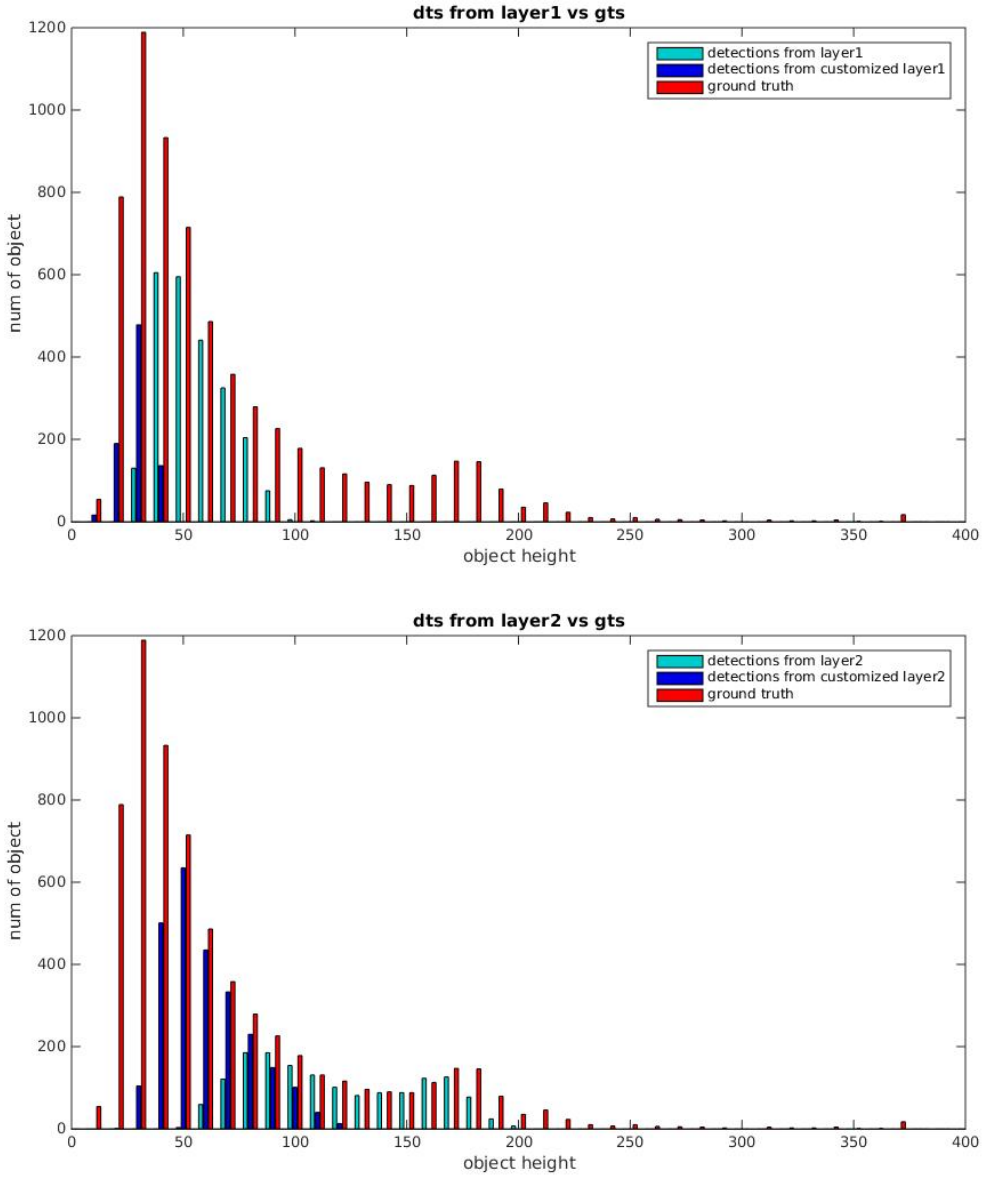

Figure 3.12: Distribution of detections from feature layer 1 and 2. (dts - detections, gts - ground truths)

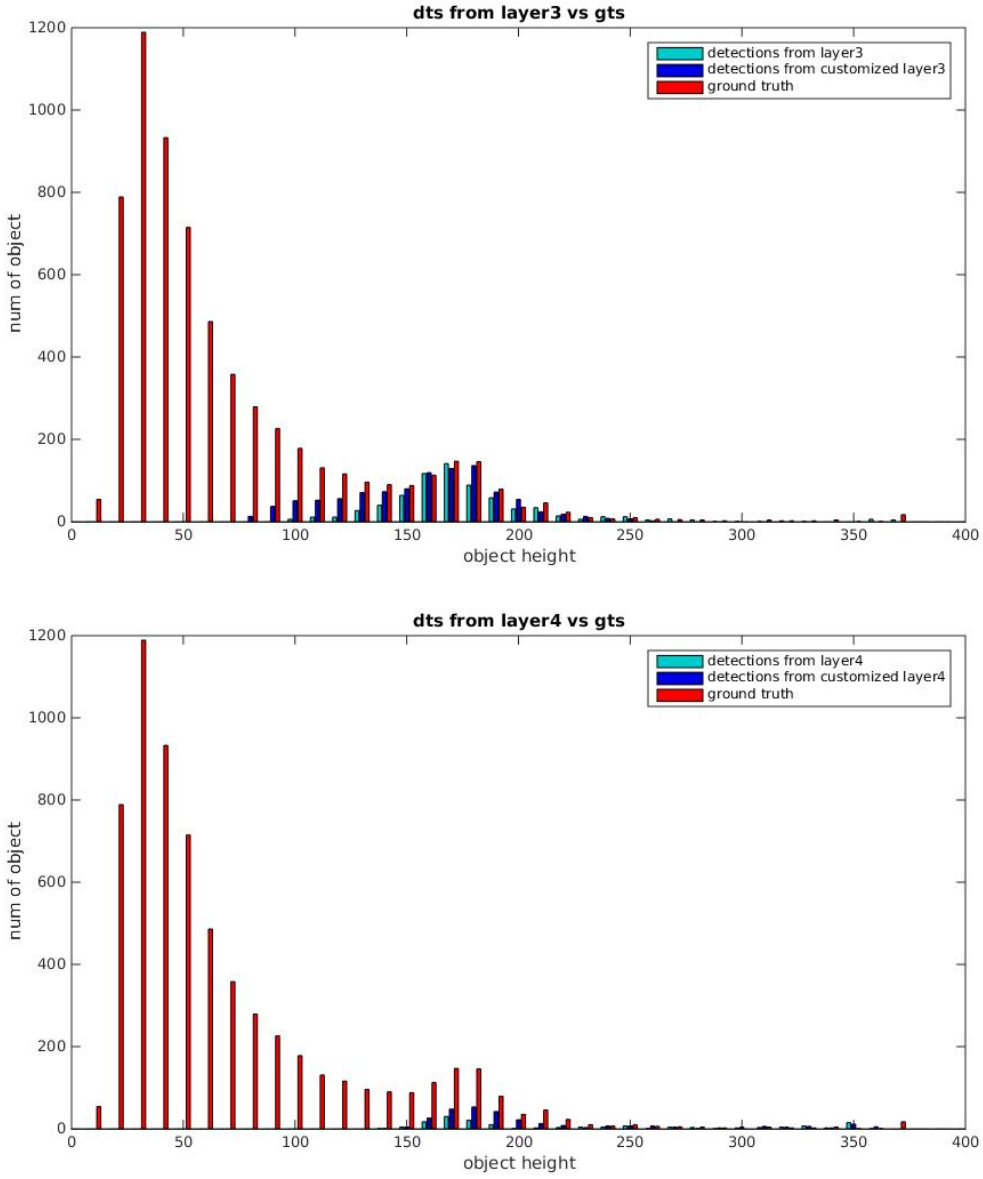

Figure 3.13: Distribution of detections from feature layer 3 and 4. (dts - detections, gts - ground truths)

Chapter 4

Two-Stream Convolutional Network

In this chapter, we present our SSD model with two-stream architecture. Details about data pre-processing and training setup are also explained. Experiment results are shown at the end of this chapter.

4.1 Two-Stream Network

While improving the model's accuracy and detection rate, we notice that the features of small and large objects are different. To further solve small object detection problem, we propose a two-stream SSD model with an extra stream especially for detecting small objects (see Figure 4.1).

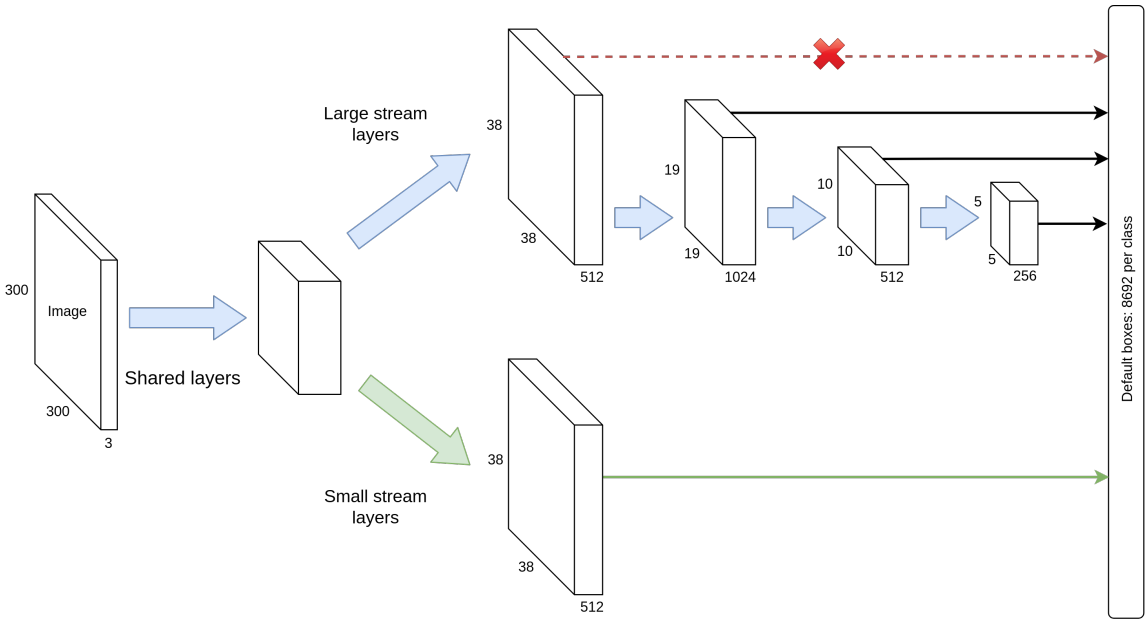

Figure 4.1: Two-stream SSD. Besides the customized 4-feature-layer network, another stream (in green) is added especially to generate feature maps for small object. Given a 300×300 image, the original 38×38 layers in upper stream is replaced by the one in lower stream so that features of small and large objects won't interfere with each other. Small object features are now extracted by the small (object) stream while the medium and large object features are handled by the large (object) stream. The amount of default boxes generated remains the same.

Instead of sharing the features extracted from target objects of different sizes, we add the second network stream to extract small object features specifically, while the medium and large objects are well handled by the first (original) stream. This architecture is inspired from the high accuracy achieved by only using the first and second feature extraction layer.

The extra stream does not increase the depth of the network but the width. It turns out that the size of the two-stream model is only about 1.5 times bigger than the original one but the mAP on KITTI dataset is improved significantly by over 3%.

4.2 Dataset pre-processing

In total, there are 7481 images with object annotations in KITTI 2D object detection dataset. For local training and evaluation purpose, all those images are separated into a training set with 5984 images and a validation set with 1497 images.

The SSD model needs to be fed with images of the same size and the size of original images in KITTI are about the same but not exactly. Therefore, before preparing the training dataset, all images are resized to 1200×350 and all annotations are also calibrated with them. There are 9 classes in KITTI but here we merge "Car", "Van" into one category as they look simiar. The other classes "Truck", "Pedestrian", "Person_sitting", "Cyclist", "Tram", "Misc" and "Dontcare" are set as background. We train our model as a single class detector for cars but the implementation can be easily transferred to multi-class detection.

The base network we use is pre-trianed on ImageNet [27] and the mean RGB values 123, 117, 104 are subtracted. Another data augmentation we follow from original paper is random crop. We implement "zoom in" and "zoom out" operations on the training samples and randomly crop the images to generate more training data.

The pre-processing for Caltech Pedestrian is the same except for there is only one class - pedestrian. Only annotations with label "person" are preserved for training and the others like "person_fa (person far away)", "person? (probably person)", "people (a group of people)" are simply discarded.

4.3 Training setup

For training step, the batch size is set to 4 and training is iterated for 120 epochs. The learing rate is 0.004 and the momentum is 0.9. To achieve better result, we refactor the learning rate after 80 epochs by 0.1 that chages the learning rate to 0.0004. We follow the setting from original paper and set the NMS threshold to 0.45 and overlap threshold to 0.5.

The default box ratios for all our experiments are inherited from original implementation (Table 4.1).

feature layer	default box ratios
1	1, 2, $\frac{1}{2}$
2	1, 2, $\frac{1}{2}$, 3, $\frac{1}{3}$
3	1, 2, $\frac{1}{2}$, 3, $\frac{1}{3}$
4	1, 2, $\frac{1}{2}$, 3, $\frac{1}{3}$
5	1, 2, $\frac{1}{2}$
6	1, 2, $\frac{1}{2}$

Table 4.1: Ratios of default boxes on each feature map.

The two sub-networks in SSD two-stream are specially trained on large and small objects separately. The shared layers are initialized using network trained on large objects.

4.4 Detection results

In this section, we present the experiment results our model and parameter settings. All computations are done on an Intel i7-6800K 3.4GHZ PC with 32 GB RAM and two NVIDIA TITAN X video cards with 12 GB memory. The main framwork and programming tools we use are MXNet 0.10.1 and MATLAB R2014b.

KITTI On the first feature map, there are 4 default boxes (with different sizes and ratios) on each position. Given a 1200×350 input, the original SSD model will generate $4 \times 22 \times 75$ default boxes in total on the first feature map. Likewise, it has $6 \times 22 \times 75$ default boxes on the second feature map, $6 \times 11 \times 38$ on the third, $6 \times 6 \times 19$ on the fourth, $6 \times 3 \times 10$ on the fifth and $4 \times 2 \times 5$ on the sixth. First scale is the smallest therefore most small object proposals are generated from the first feature map. The size of default boxes on layer 5 and 6 are the largest and even too large for any regular vehicles on image. The number of generated default boxes remains the same for each layer on our model, but the box sizes are different (Table 4.2). Original SSD set the maximal scale at 0.9 and the minimal scale as 0.1. Here we extend the minimal to 0.03 while keep the maximal scale 0.9. According to the calculation from 3.1.3, we can get the size of default boxes on each layer. Take the first layer as an example, the size of default boxes is [.1, .141] in original SSD but is

[.03, .0548] in our Two-Stream SSD. That enables the our model to generate proposals for smaller objects on the first layer. Last two layers are removed in our Customized SSD and Two-Stream SSD.

Table 4.2: Comparison of default box sizes of each layer between SSD, Customized SSD and Two-Stream SSD. [s_1, s_2] in the table represent the minimal scale s_1 and maximum scale s_2 of the default boxes on that layer (as discussed in Section 3.1.3). For original SSD, the smallest scale is .1 on the first layer and the largest scale is .96 on the sixth layer. In our customized SSD, we extend the smallest scale to .03 and truncate last two layers as their scales are too large to fit any object. As for our two-stream SSD, we use the same default box scales as the customized SSD and move the first layer to another stream. Overall, customized SSD has four feature extraction layers. In the two-stream model, default boxes are generated from layer 2, 3 and 4 on the first stream (for medium and large objects) and layer 1 on the second stream (for small objects). Dashes ("-") mean there's no such layer in this model.

model	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6
SSD	[.1, .14]	[.2, .27]	[.37, .44]	[.54, .62]	[.71, .79]	[.88, .96]
Customized SSD	[.03, .05]	[.1, .17]	[.3, .38]	[.5, .59]	-	-
Two-Stream SSD	-	[.1, .17]	[.3, .39]	[.5, .59]	-	-
	[.03, .05]	-	-	-	-	-

We mainly compare four models here: SSD, SSD customized, SSD Two-Stream* (sharing one stage), SSD Two-Stream** (sharing two stages). The mAP results of those models are shown in the Table 4.3.

Table 4.3: mAPs with different models and training samples. All mAP are evaluated on KITTI test set. * means two streams share one stage, ** share two stages.

Model	mAP
SSD	78.565%
SSD customized (ours)	85.614%
SSD two-stream* (ours)	**88.624%**
SSD two-stream** (ours)	88.546%

The best result is achieved by SSD Two-Stream* (sharing one stage) and it outperforms the original SSD by 10% mAP. We also generate the precision-recall curve for comparison (Figure 4.2). The performance of sharing one or two stages don't make much difference.

For convenience, we use and refer the Two-Stream SSD sharing one layer as Two-Stream SSD in the following sections.

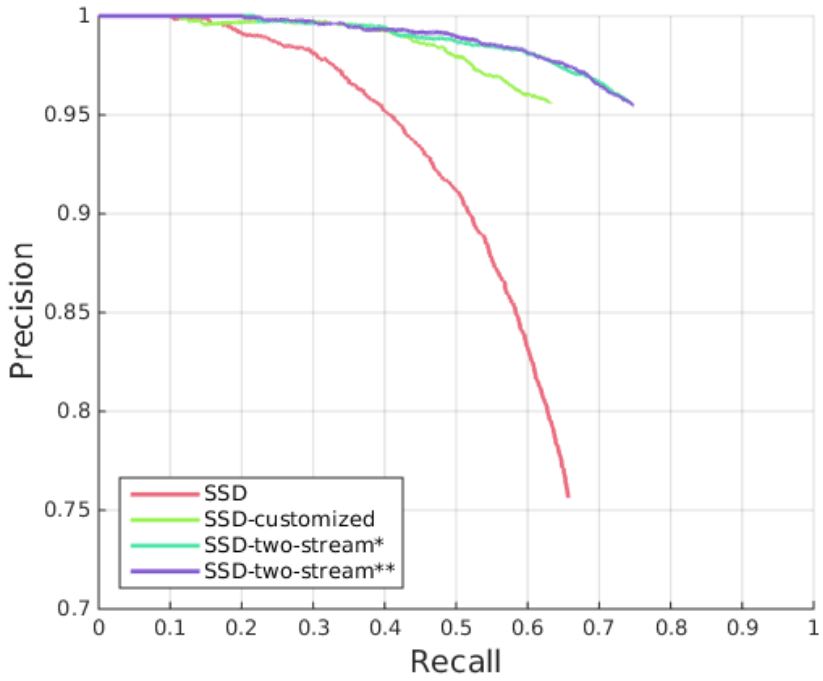

Figure 4.2: Precision recall curve for SSD, SSD customized, SSD two-stream* (sharing one stage), SSD two-stream** (sharing two stages) models.

Finally, we trained our Two-Stream SSD model on all 7418 training images and evaluated on the KITTI server. We also trained the original SSD using same hyper-parameters and settings for comparing purpose. The KITTI testing set separates the objects into three categories depending on their difficulty of detection: Easy, Moderate and Hard. The evaluation result (Table 4.4 and Figure 4.3) shows our Two-Stream model achieves much higher accuracy detecting Moderate and Hard objects than the original SSD, while having almost similar performance on Easy objects detection. The slightly descending of performance on Easy objects is caused by the first few sharing layers between two streams and less default box proposals for large objects since configuration changed.

Table 4.4: SSD vs SSD Two-Stream from KITTI online evaluation of car detection.

	Easy	Moderate	Hard
SSD	**85.66%**	69.57%	61.27%
SSD Two-Stream (Ours)	84.68%	**77.59%**	**69.21%**

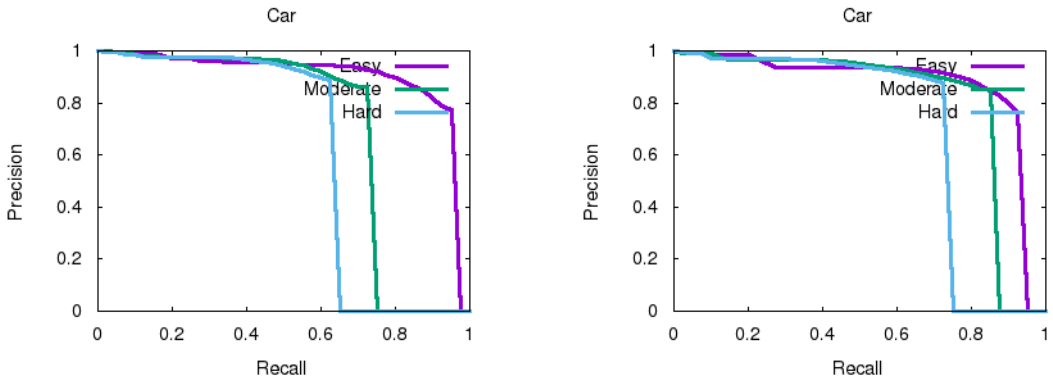

Figure 4.3: SSD (left) vs SSD Two-Stream (right) from KITTI online evaluation of car detection.

Caltech Pedestrian It's convienent to conduct experiments and tests on Caltech Pedestrian as it provides the demo and evaluation scripts. We also trained the two-stream SSD and the original SSD on Caltech Pedestrian dataset. Here we show some demos (Fig 4.4) inferenced by those two network, which share the same training configurations and procedures.

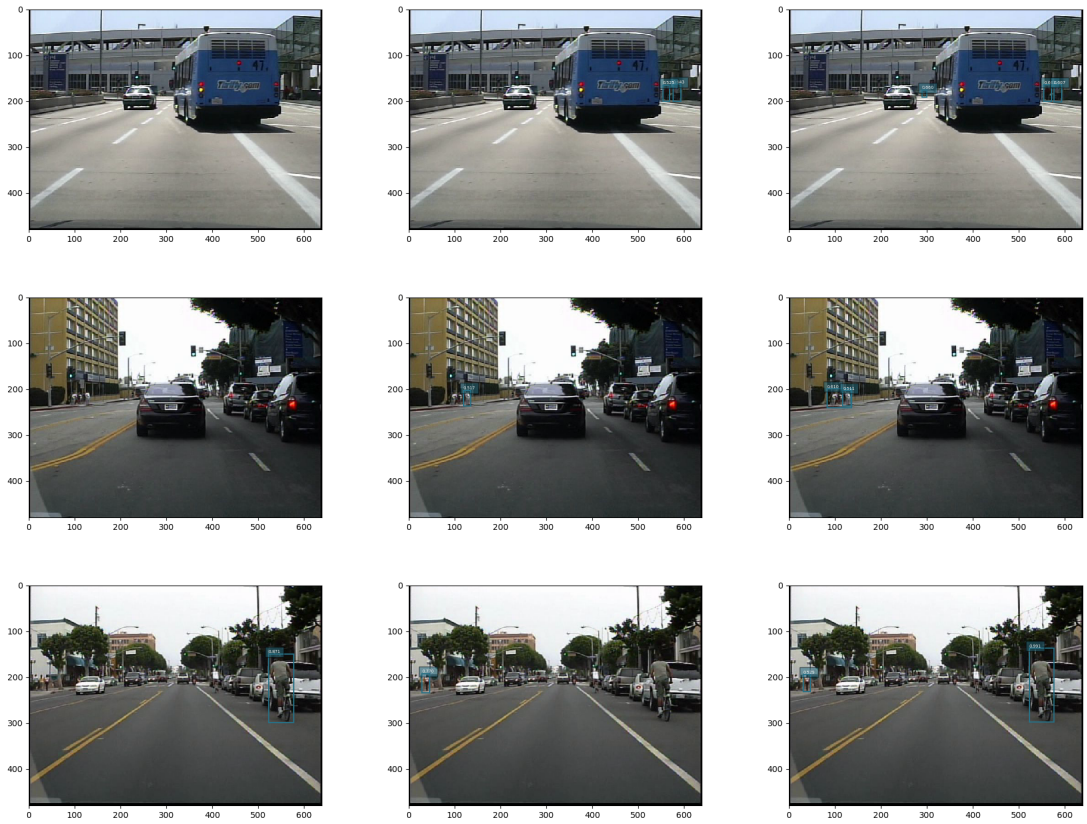

Figure 4.4: Demo on Caltech Pedestrian from SSD(left), SSD Customized(middle) and SSD Two-Stream(right). We can see that the original SSD and SSD Customized are good at detecting large and small objects respectively, while SSD Two-Stream takes advantages of both.

Still, we mainly tested our customized model and two-stream performance on this dataset. We followed the same statistical analysis procedure as we did on KITTI and found the majority of target objects in Caltech Pedestrian have small to medium size. The last two layers of original SSD is again not used due to no contribution to the final detection result, so we simply removed the last two layers and adjust the minimal and maximal scale to form our customized model. However, to construct our two-stream model, since the vast majority of target objects' height are around 50 (as shown in Figure), we construct the large stream using first 4 layers from original model (instread of 3 in KITTI) to generate

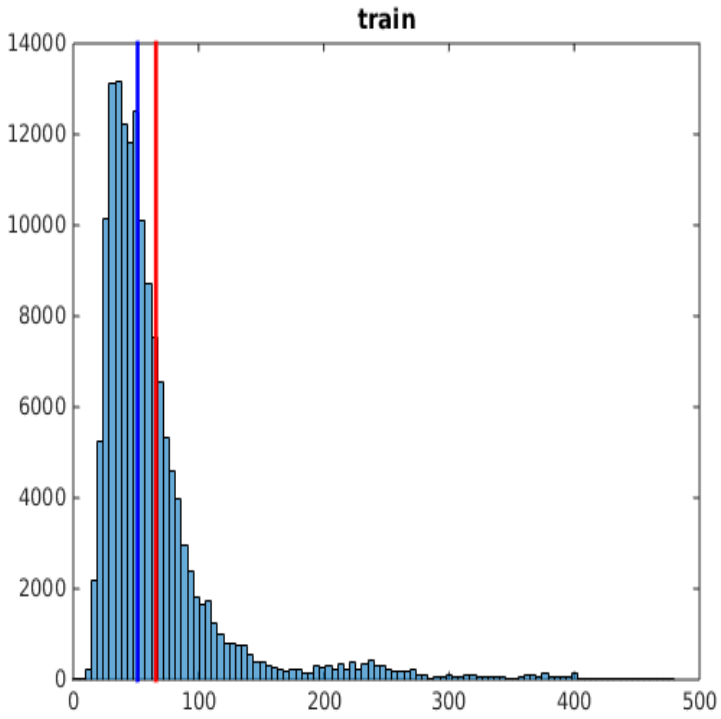

Figure 4.5: Target objects height distribution in Caltech Pedestrian. Red line is the mean (65.9) while blue line is the median (51.3).

more proposals for potential objects with height around 50. The small stream still only consists the first layer of customized model.

Overall the multibox layers setup are list in Table 4.5:

Table 4.5: Multibox layer setups for Caltech Pedestrian dataset. The large stream in two-stream model here uses same configuration of first four layers in original SSD, in order to get more multibox proposals for the small and medium sized target objects. Customized SSD only has four layers. In the two-stream model, default boxes are generated from feature layer 1, 2, 3, 4 on the first stream (for medium and large objects) and feature layer 1 on the second stream (for small objects). Dashes ("-") mean there's no such layer in this model.

model	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6
original SSD	[.1, .14]	[.2, .27]	[.37, .45]	[.54, .62]	[.71, .79]	[.88, .96]
customized SSD	[.03, .05]	[.1, .17]	[.3, .39]	[.5, .59]	-	-
two-stream SSD	[.1, .14]	[.2, .27]	[.37, .45]	[.54, .62]	-	-
	[.03, .05]	-	-	-	-	-

To evaluate the model performance more specifically, Caltech gives a set of experiments with different object height and visibility ranges. For example, we can evaluate our model only on small or large objects with no occlusion or heavy occlusion. In total, we chose 6 experiments with different settings of size and visibility to test our model's performance, which are All, Reasonable, Far, Medium, Near and Large. Details about the experiment settings can be found in Table 4.6.

Table 4.6: 6 experiments of different object size and visibility given input image size 480x640.

	Size	Visibility
All	$[20,\infty)$	$[.2,\infty)$
Reasonable	$[50,\infty)$	$[.65,\infty)$
Far	$[20,30)$	$[.2,\infty)$
Medium	$[30,80)$	$[.2,\infty)$
Near	$[80,\infty)$	$[.2,\infty)$
Large	$[100,\infty)$	$[.2,\infty)$

As shown in Figure 4.6, given an 600×800 image as input, the original SSD model has an average miss rate of 64% on All experiment set while our two-stream model improves it to 62%. The Customized model achieves highest accuracy on Small (Far) set but in general not so well as it has much less proposals than the other two. The fact that our Two-Stream model outperforms the original is mainly benefited by its improvement on Far(small) and

Medium objects detection. And the reason for its unsatisfactory performance on Reasonable set mainly comes from low accuracy on Near and Large object sets. This behavior is similar with what we found on KITTI dataset and it's caused by the sharing layers of the large stream, which is reponsible for detecting large objects. We tried to train our model using different input size (480×640, 600×800 and 720×960) and we noticed the larger input size will benefit the model to achieve higher accuracy (Figure 4.7). We also did some experiments on the Caltech Pedestrian New dataset, which uses Caltech Pedestrian images with cleaner but much fewer annotations (Figure 4.8).

With those experiments, we can conclude that our Two-Stream model outperformes the original SSD model significantly on detecting small to medium sized objects in KITTI and Caltech Pedestrian and improves the overall accuracy. The improvement came from the extra stream, which has clearer small objects feature, as well as the geometry analysis, which were used to adjust the parameters of multibox layers.

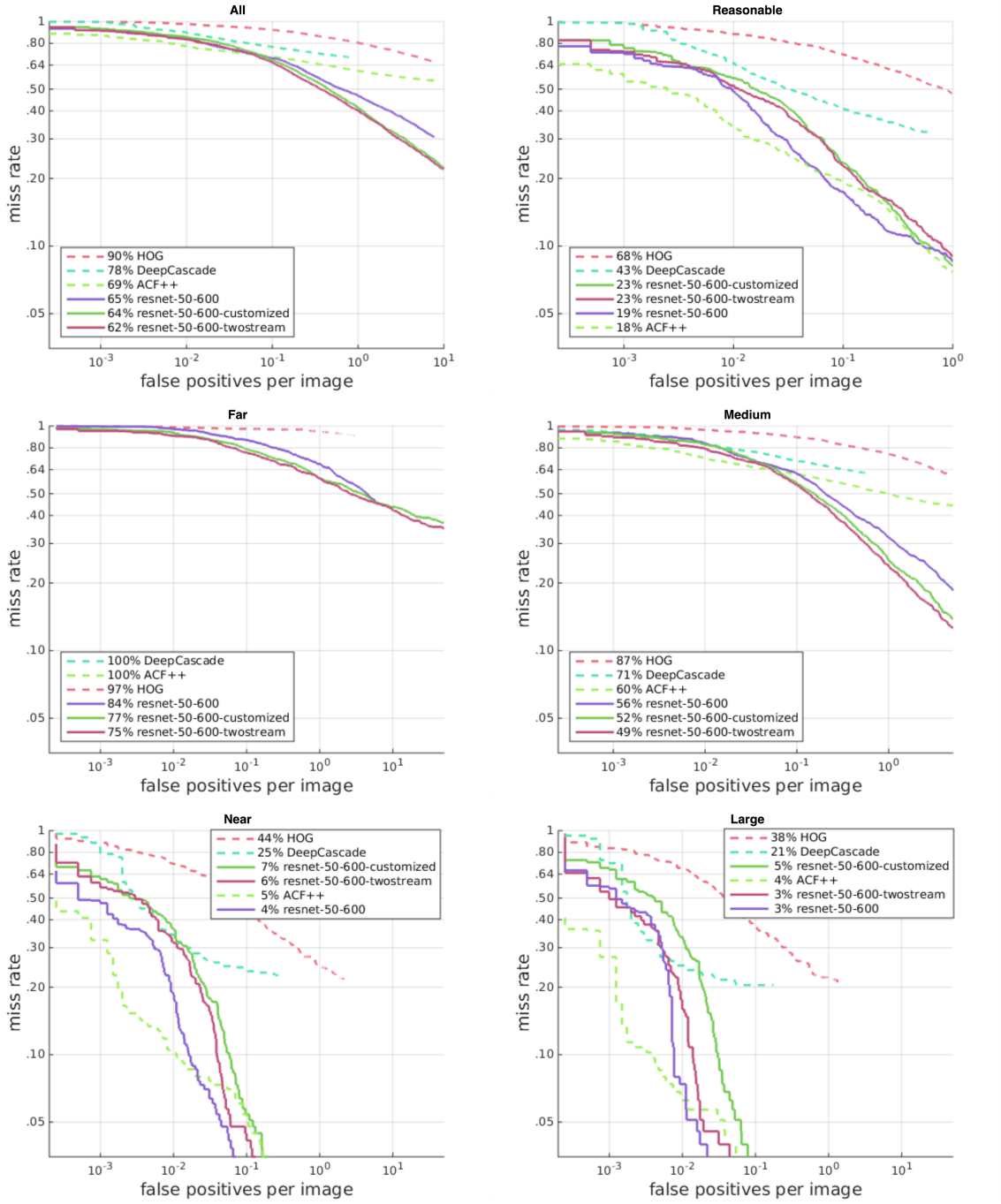

Figure 4.6: Caltech Pedestrian FPPI curve and comparison between ResNet50, ResNet50-customized and ResNet50-two-stream given input size 600×800. From top to bottom, 5 figures are respectively (1)all, (2)far, (3)medium, (4)near, (5)large. Original SSD model has an average miss rate of 65% while our revised two-stream model has 62%. The customized model achieves best at detecting far objects but in general not so well as it has much less proposals than the other two. The two-stream model achieves best performance at detecting all objects and medium objects. Curves with 100% are omitted on the plots.

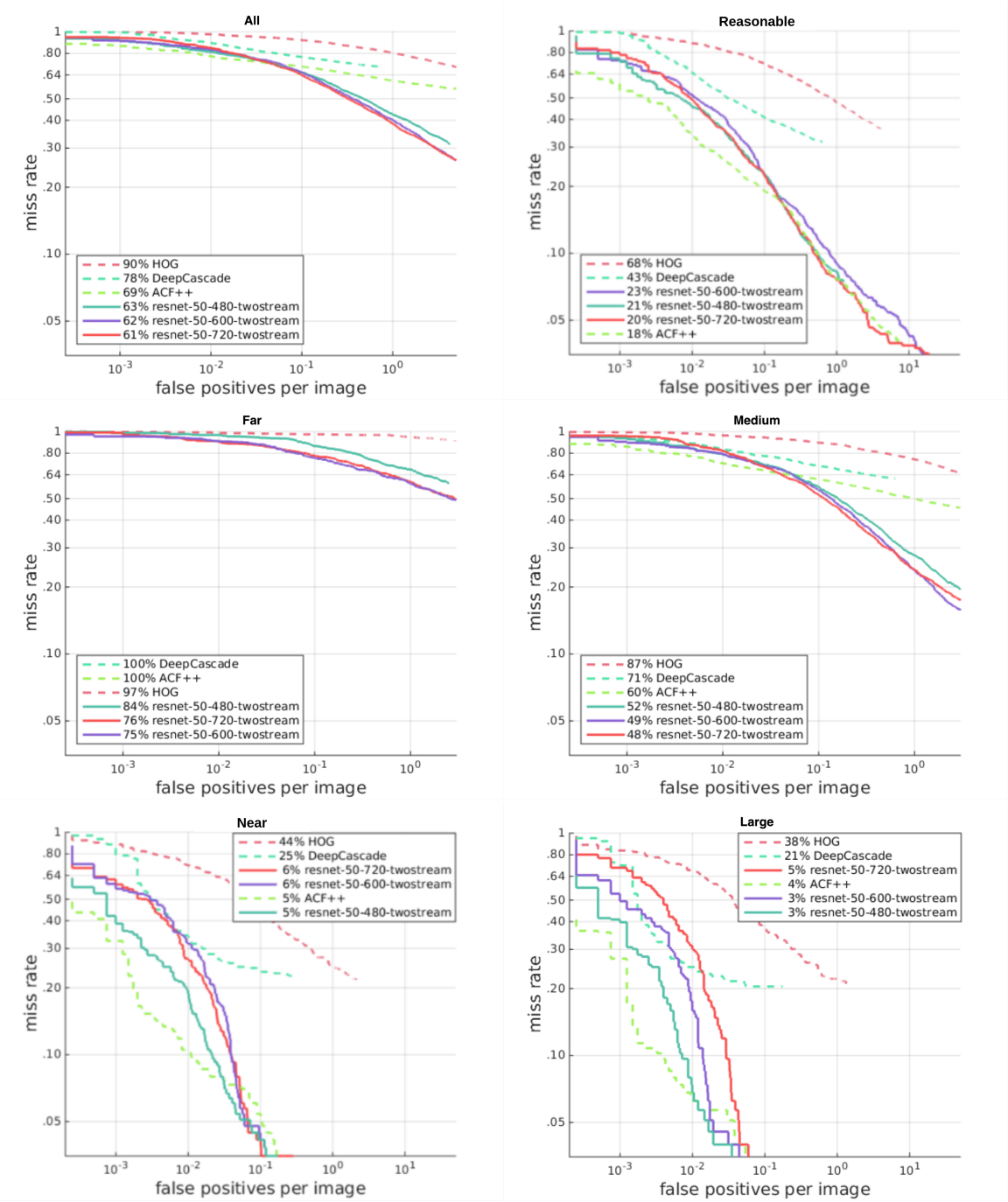

Figure 4.7: Caltech Pedestrian FPPI curve and comparison of our proposed ResNet50-two-stream between input image size 480x640, 600×800 and 720x960. Overall, we notice that better performance can be achieve using large image as input. Curves with 100% are omitted on the plots.

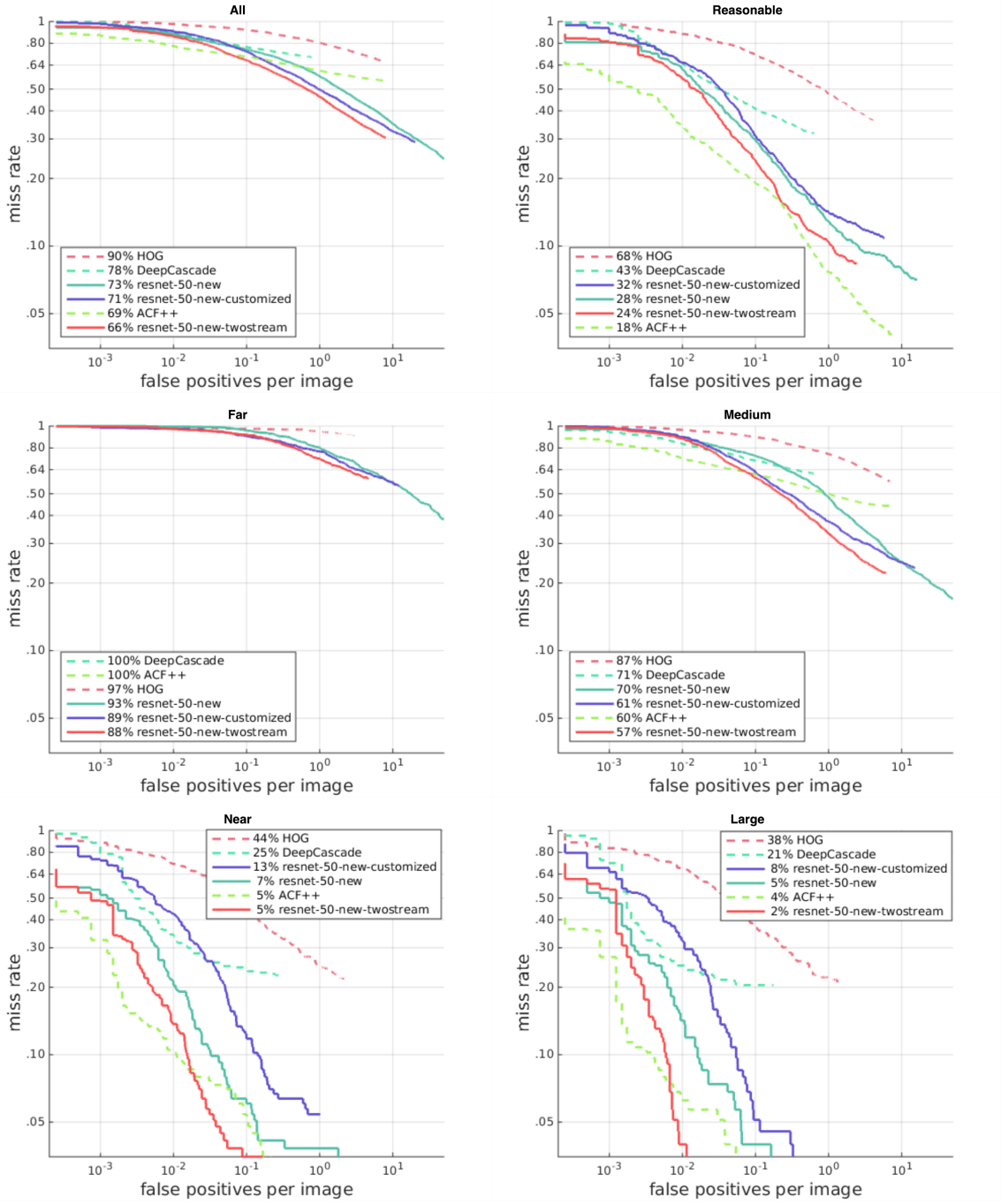

Figure 4.8: Caltech Pedestrian New FPPI curve and comparison between ResNet50, ResNet50-customized and ResNet50-two-stream given input size 480x640. Our Two-Stream model still achieves overall better performance with clearer annotations but much fewer training samples. Curves with 100% are omitted on the plots.

Chapter 5

Summary and Conclusion

5.1 Summary

We present several object detection models based on SSD architecture and test their performance on KITTI and Caltech Pedestrian dataset. The parameters of feature extraction layers are optimized after analysing the training dataset. Specifically, an SSD model with two streams are proposed to increase accuracy and detection rate especially for small objects. This model has the same depth as the original SSD and its first few layers can be shared between two streams to reduce the model size. Our Two-Stream SSD model outperforms the original SSD model significantly on accuracy and achieves competitive results on KITTI and Caltech Pedestrian dataset.

5.2 Limitations

The training procedure for the two stream network is more complicated than the original SSD, since we have to pretrain each stream on large and small objects separately. Even the first two stages of network are shared, about 35% more memory consumption is introduced by the extra stream. This model needs to perform scene geometry analysis manually before training in order to find best configuration of network. Aspect ratio of default boxes are not optimized based on target objects' characteristics. For example, most of the car objects are flat (with larger width than height) but the vertical default boxes (with larger height

than width) are still generated but having very low possibility of match. Occluded objects are treated as fully visible during training for simplicity, however they would confuse the network by such noise.

5.3 Future works

The aspect ratio of objects can also be analysed before training. Those information can be used to reduce the amount of default boxes thus further increase the inference speed. According to our experiments, the two-stream networks have approxiamately the same performance regardless of the amount of their shared layers. More experiments can be conducted to explore the limit of two-stream model without a significant drop of performance. Geometrical distribution of objects in training samples can also be used to further reduce the size of the model. This problem can be solved by self-adaptive SSD network which automatically adjust feature extraction layer configurations during training. Deeper network such as ResNet-101 could be used to further improve accuracy but with a trade-off of inference speed. Feature extracted from large object stream can be used to provide contextual information for the small object stream like [13].

Bibliography

[1] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit S Ogale, and Dave Ferguson. Real-time pedestrian detection with deep network cascades. In *BMVC*, volume 2, page 4, 2015.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[3] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[4] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[8] Navneet Dalal and Bill Triggs. Inria person dataset. *Online: http://pascal. inrialpes. fr/data/human*, 2005.

[9] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2012.

[10] Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, and Larry Davis. Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 953–961. IEEE, 2017.

[11] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE Press, June 2008.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[13] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[16] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[20] Huieun Kim, Youngwan Lee, Byeounghak Yim, Eunsoo Park, and Hakil Kim. On-road object detection using deep neural network. In *Consumer Electronics-Asia (ICCE-Asia), IEEE International Conference on*, pages 1–4. IEEE, 2016.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Jianan Li, Xiaodan Liang, ShengMei Shen, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Scale-aware fast r-cnn for pedestrian detection. *IEEE Transactions on Multimedia*, 2017.

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[24] Eshed Ohn-Bar and Mohan M Trivedi. To boost or not to boost? on the limits of boosted trees for object detection. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 3350–3355. IEEE, 2016.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[28] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[29] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Xudong Sun, Pengcheng Wu, and Steven CH Hoi. Face detection using deep learning: An improved faster rcnn approach. *arXiv preprint arXiv:1701.08289*, 2017.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[33] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[34] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[35] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. *arXiv preprint arXiv:1704.03414*, 2017.

[36] John H Wensley, Leslie Lamport, Jack Goldberg, Milton W Green, Karl N Levitt, Po Mo Melliar-Smith, Robert E Shostak, and Charles B Weinstock. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.

[37] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5525–5533, 2016.

[38] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. Is faster r-cnn doing well for pedestrian detection? In *European Conference on Computer Vision*, pages 443–457. Springer, 2016.