Understanding and affecting the accuracy/speed trade-off of object detectors for self-driving cars

Xurui Yan

Master of Science

Informatics

School of Informatics

University of Edinburgh

2019

Abstract

Vision-based self-driving cars take advantage of cameras combined with object detection engine to detect cars, persons, traffic lights, etc in the environment. This application requires the object detector to have high accuracy and low latency in order to avoid accidents.

The main aim of this project is to investigate different ways to trade-off between accuracy and speed, including modifying both the model and input images, for a state-of-the-art object detector, YOLO. We evaluated the performance of YOLO on the Berkeley DeepDrive (BDD) dataset while varying the number of detection layers and anchors (i.e., bounding box priors), image resolution and applying image enhancement techniques. Many of our findings have not been reported elsewhere. We found that additional detection layers can improve the detection accuracy significantly because small objects are detected on a fine-grained feature map. The performance is quite robust to the number of anchors generated by clustering methods like K-Means. Image resolution is one of the most effective ways of affecting the accuracy/speed trade-off. Higher resolution usually yields better performance but too high resolution may also lead to worse performance. Among the two image enhancement techniques we applied, deblurring should help provided that a good deblurring method exists while contrast enhancement, surprisingly, does not help.

Part of the aim is to compare YOLO with RetinaNet which can achieve much higher accuracy on the COCO object detection competition. Our experiment failed to demonstrate the advantage of RetinaNet on BDD dataset in either accuracy or speed. Since fair comparison is very difficult, some suggestions are discussed when selecting the right object detection method.

Our final YOLO model that combines several optimization techniques we have learned in this study achieves 25.75 mAP@0.75 and 43 FPS and surpasses the third place on the WAD Road Object Detection challenge leaderboard, indicating the significance of our work.

It is hoped this research will contribute to a deeper understanding of the accuracy/speed trade-off of object detectors and provide practitioners who are building object detection systems for self-driving cars with some insights into tuning YOLO to achieve high accuracy while still running in real time.

Acknowledgements

I would like to sincerely thank my supervisor, Professor Boris Grot, for his excellent guidance and feedback throughout the dissertation as well as his GPU resource without which this project could not be completed. I am very grateful to have had the opportunity to learn so much about both research and writing from him in the past few months.

I would also like to express my gratitude to my personal tutor Tom Spink for reading my dissertation and providing some comments, to Artemiy Margaritov for his insightful suggestions and availability to help, to Jamie Norris for sharing information during this project.

Finally, I would like to thank my family and friends for their unwavering support.

Table of Contents

Chapter 1

Introduction

In recent years, there has been an increasing interest in self-driving cars which are expected to revolutionize the way of mobility in the future even though they are still largely under experiment. Because of the great success in computer vision, a vision-based self-driving car which senses the environment using cameras becomes the latest research trend [1]. Therefore, object detection is the key component of a self-driving car to detect pedestrians, vehicles, traffic lights and other objects of interest around the car from the vision information captured by the camera.

However, existing object detection methods remain a bottleneck in such applications due to the real-time processing requirements. Recent research [1] suggests that the latency of the self-driving system should be less than 100ms in order to react fast enough to avoid traffic accidents. This speed requirement presents challenges to modern object detection algorithms based on neural networks which are very computationally intensive. Driven by object detection competition such as COCO challenge, recent advances in object detection have put too much emphasis on accuracy and only a few of them take speed into account [2]. Faster R-CNN [3] is the first attempt towards real-time object detection and, later, one-stage object detectors, such as YOLO (You Only Look Once) [4] and SSD [5], reach the real-time milestone with relatively low accuracy. Another one-stage detector RetinaNet [6] shares many similar architecture designs with previous detectors but features a novel focal loss function. Its accuracy on COCO dataset surpasses previous state-of-the-art detectors such as Faster R-CNN and DSSD [7]. Recently, the third and latest version of YOLO, YOLOv3 [8], has made a huge improvement in accuracy and is still fast enough to be run in real time, which makes it the most suitable candidate for object detection engine in the self-driving scenario. In the remainder of this dissertation, the term YOLO will refer to YOLOv3

specifically. However, previous research [1] shows that its latency can not meet the design constraint (i.e., 100ms) of self-driving systems when using high-res images as input even if it is running on the GPU platform. Since it is not easy to improve the speed for existing methods or hardware platforms, there is an urgent need to select the appropriate configurations for YOLO that balance the detection accuracy and throughput in a real time application like self-driving system.

Extensive research [8, 6, 9, 2] has shown that image resolution has a significant impact on the accuracy and speed of object detection. In particular, a common way to trade accuracy for speed is reducing the image size before feeding it into the network. However, little is known about how other aspects can influence the accuracy/speed trade-off.

From the perspective of a practitioner, both the model configurations and data preprocessing can be taken into consideration to achieve the right accuracy/speed balance for a specific application. One solution is to focus on the model itself which can be configured to achieve higher accuracy or lower latency. Most one-stage object detection methods, including YOLO, predict objects at different scales in the feature pyramid based on a number of anchors (i.e., predefined bounding boxes). Evidence [10] suggests that multi-scale detection is among the most important architecture designs for detecting objects of different sizes. YOLO detects objects at three different scales by default and additional scales may help to improve the accuracy. Another key design concept is anchor and it was shown that anchor density can affect the performance by covering different object scales and aspect ratios [6]. The other solution is adjusting the input images during inference. Since YOLO can be trained or tested at different resolutions, resizing the input images offers an easy way to trade-off between accuracy and speed. In addition to resizing, another important data preprocessing technique is image enhancement, such as deblurring and contrast enhancement, which can significantly improve the quality of images for human eyes and might benefit the performance of object detection as well.

The primary objective of this project is to investigate various ways to affect the accuracy/speed trade-off for a modern object detection system, specifically YOLO. To this end, we evaluate the performance of YOLO by measuring the accuracy and speed as a function of different detection scales, anchor density, image resolution, whether performing image enhancement or not on the large-scale Berkeley DeepDrive (BDD) dataset [11] developed for the purpose of self-driving. Other choices of trade-off such as ensemble or multiple inference[1] are not considered because they are too

computationally expensive to be used in a real-time application.

Since RetinaNet may also be a competitive candidate for our application, the second objective is to compare the performance of YOLO with RetinaNet on BDD dataset and decide which is preferable. To demonstrate the important implications of this study for practitioners, the last objective is to train a model that is expected to be more accurate and also faster than existing methods for the BDD road object detection challenge by combining those improvements we have learned in this project.

Our main contributions include:

- This study is the first to explore various ways of trade-off between accuracy and speed for YOLO and will contribute to a better understanding of the accuracy/speed trade-off for an object detector.
- We present a comparison between YOLO and RetinaNet and our findings will provide insights into selecting the right object detection system.
- We show how several optimization techniques are combined together to train an object detection system that can achieve very good accuracy (25.75 mAP@0.75 measured on BDD test set) and in the meanwhile, run in real time (43 FPS).

The remainder of the document is organized as follows. Chapter 2 describes the background which will facilitate the understanding of our work. Chapter 3 is concerned with the dataset used for this study. Chapter 4 presents the details of YOLO. Chapter 5 describes each experiment undertaken in details and presents results with interpretation. Chapter 6 summarizes what have been achieved in this project, acknowledges the limitations and suggests ideas for further research.

[1]The phrase 'multiple inference' will be used in this dissertation to refer to techniques that pass an input image through a model multiple times and than merge the results during test. E.g., multi-scale and multi-crop inference.

Chapter 2

Background

This chapter shows the broader context of object detection and then introduces the problem in a particular application that our research will be addressing. The purpose is to familiarize readers with the background in this field, such as the evaluation metric, and also to have a better understanding of the timeliness of our work.

2.1 Evaluation Metric

Object detection is a multitask problem that identifies objects in an image by classification and localization using bounding boxes. For each object detected from the input image, the detector outputs its category, the coordinates of the bounding box and a confidence score.

Mean average precision (mAP) is the most common metric for measuring how well object detectors locate and classify objects in the image. To understand this measurement, it is necessary to understand Intersection over Union (IoU) first. As illustrated in Figure 2.1, IoU measures the overlap between two bounding boxes and high IoU means two bounding boxes are highly overlapped with each other.

Average precision (AP) is first calculated in each category as follows. For each ground truth box, a predicted bounding box is considered as true positive (TP) only if it is the closest to the ground truth box in terms of IoU and the IoU between them is

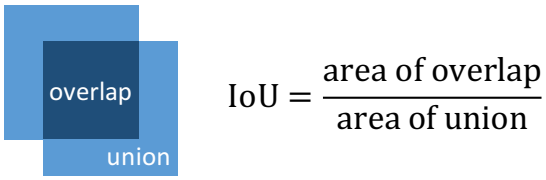
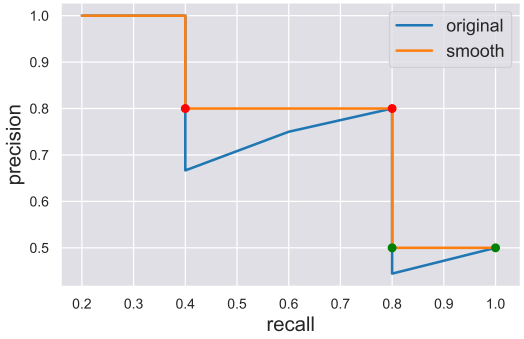


Figure 2.1: The definition of Intersection over Union.

rank	score ↓	correct?	recall	precision
1	0.99	True	0.2	1.00
2	0.91	True	0.4	1.00
3	0.87	False	0.4	0.67
4	0.85	True	0.6	0.75
5	0.80	True	0.8	0.80
6	0.64	False	0.8	0.67
7	0.51	False	0.8	0.57
8	0.32	False	0.8	0.50
9	0.11	False	0.8	0.44
10	0.09	True	1.0	0.50

(a)



(b)

Table 2.1: An example of how to calculate AP. (a) Predictions are ranked by confidence score. Precision is the proportion of TPs in predictions from the first row to current row. Likewise, recall is the proportion of TPs in all positives. This example assumes there are 5 positives. (b) The AP is the area under the smooth PR curve which is transformed from the original one.

also higher than a specified threshold (e.g., 0.5). The remaining predicted boxes are false positives (FP). Then, the predictions in all images are ranked in descending order according to the confidence score. Since each prediction has been classified as TP or FP, precision and recall are calculated at each entry in the ranked list by taking all preceding predictions into account. As shown in the example in Table 2.1a, the recall rises when going down the ranked list while precision shows a zigzag pattern: it goes down with FP but goes up again with TP. The blue line in Figure 2.1b shows the curve if we plot the precision against recall. Average Precision (AP) is the area under the precision-recall curve after smoothing out the zigzag pattern by replacing the precision at each recall with the maximum precision on the right. Since both precision and recall are between 0 and 1, AP also falls in the same range.

mAP is then calculated by averaging the AP in all categories. The mAP for the road object detection task in WAD 2018 Challenges hosted by Berkeley DeepDrive is calculated in this way at an IoU of 0.75.

Recently, researchers tend to report the COCO mAP which uses a 101-point interpolated AP. This AP calculates the average of precisions on the smoothed precision-recall curve at 101 recalls ranging from 0 to 1 with a step size of 0.01. It should be

noted that this AP is slightly different from calculating the area under the curve directly. The COCO mAP (denoted by mAP@[.5:.95]) is then calculated by averaging this AP over all categories and 10 IoU thresholds from 0.50 to 0.95 with a step size of 0.05. This mAP is often simply referred to as COCO AP in literature.

We adopt COCO mAP because it is more widely used and COCO API also offers a flexible way to analyze the relationship between the mAP and different object sizes or classes. In addition to mAP@[.5:.95], COCO API also reports mAP@0.5 (i.e., mAP at IoU 0.5), mAP@0.75 and mAP[.5:.95] for small, medium and large objects respectively. For simplicity, the term mAP alone will refer to COCO mAP@[.5:.95] and mAP always appears as a percentage without % throughout this document.

2.2 Object Detection Methods

Previously, sliding-window approach was the leading detection paradigm in which a classifier was applied on an image grid [12]. In recent years, more efficient detection methods based on deep learning have dominated this visual task. Modern object detectors can be divided into the following two categories.

Two-stage Detectors: As popularized by R-CNN (Region-based Convolutional Network) [13], this approach applies a classifier to a sparse set of region proposals that contain candidate objects. In R-CNN, the first stage extracts region proposals from the original image using a selective search method and the second stage forwards each region through a CNN followed by a classifier and a bounding box regressor. In the past few years, R-CNN has been improved in terms of speed. Fast R-CNN [14] forwards the whole image through a CNN once and then selects regions of interest (RoIs) from the feature map of the last convolutional layer instead of the original image. Faster R-CNN [3] replaces the time-consuming selective search method in Fast R-CNN with a region proposal network that shares the same CNN backbone as the detection network and reduces the inference time significantly from 2s to 0.2s. Mask R-CNN [15] is built on Faster R-CNN and can deal with instance segmentation by additionally predicting a mask for each bounding box.

One-stage Detectors: One-stage approach predicts objects by forwarding the input image through a CNN once without region proposals. SSD [5] and YOLO [8] are two of the most widely used one-stage detectors. Both use pre-defined anchors and detect objects at multiple feature maps that cater objects of different sizes. YOLOv3 [8] has made a huge improvement over YOLOv2 [16] by using a deeper backbone and

incorporating Feature Pyramid Network (FPN) [10] and its accuracy has surpassed its main competitor, SSD. Although they can run at real time, their accuracy is still not comparable with that of two-stage methods. RetinaNet [6] combines the ideas of previous dense detectors, such as anchor and FPN with a novel focal loss and improves the accuracy further, surpassing most existing two-stage detectors. RetinaNet is composed of a backbone such as ResNet-101 [17] with FPN and two task-specific subnetworks: one for classification and the other for bounding box regression. However, the novelty of RetinaNet is that it uses a new focal loss to address the background-foreground class imbalance in one-stage detectors due to the fact that most pixels are in background (i.e., negative) and make learning for positives inefficient.

One of the key challenges for object detection is scale variation because the size of objects in the image usually varies in a wide range (e.g., a car is very big near the camera but very small in the distance) but CNN has difficulty handling objects of different sizes especially those very small or large objects.

An intuitive solution is to resize the input image to have different scales, forming an image pyramid. SNIP [18] adopts this approach and selectively trains on objects of a fixed range of size at each image scale. This detector is too slow because its scale-aware nature requires to process an image at different scales to detect objects of different sizes. SNIPER [19] improves the speed of SNIP by only processing context regions around ground truths generated by a region proposal network in stead of the full image. Based on Faster R-CNN with a ResNet-101 backbone, this multi-scale training schema obtains an mAP of 47.6 on the COCO object detection task, namely the test-dev set. AutoFocus [20] improves the inference speed of SNIPER by only processing regions (called FocusChips) that are likely to contain objects at finer scales instead of using RPN on an entire image pyramid.

A more efficient alternative is feature pyramid. FPN [10] augments a standard CNN with a top-down pathway and lateral connections such that finer-grained and local information from low level feature maps is combined with coarser and global information from high level feature maps. This technique is shown to improve multi-scale object detection and is now widely used in the literature. Both RetinaNet and YOLO adopt this approach while SSD approximates this idea by detecting objects from multi-scale feature maps without the top-down pathway.

TridentNet [21] adopts a novel network structure to handle the scale variation problem. TridentNet uses dilated convolutional layers popularized by [22] with shared parameters but different dilation rates in multiple parallel branches. This method com-

bined with ResNet-101 as backbone achieves an mAP of 42.7 on COCO test-dev.

2.3 Object Detection for Self-driving Cars

A self-driving car, also known as autonomous vehicle, is able to drive by itself without human manipulation. This emerging and exciting application has attracted a lot of interest from both industry and academic world. For instance, Google has invested a lot in developing self-driving cars in the past few years.

A self-driving car is equipped with sensors and self-driving systems to detect and navigate in the environment. Currently, the industry tends to build vision based self-driving systems which employ cameras as sensing devices because cameras are relatively cheap and can provide rich information about the surroundings [23]. Therefore, object detection algorithms are essential for a vision based self-driving system to detect objects of interest, such as vehicles, people and traffic lights, from images taken by the camera.

Lin et al. pointed out that the self-driving system needs to be able to process the traffic information in real time and react to the environment faster than human drivers to avoid accidents, which suggests the processing time needs to be less than 100ms. However, this speed constraint presents a challenge for most object detectors. One reason is that current state-of-the-art object detection algorithms are usually based on deep convolutional networks which are computationally intensive. The other reason is that high-res cameras are used to keep as many details in the environment as possible. It is therefore important to find a balance between accuracy and speed for this real-time application.

2.4 Accuracy/speed Trade-off

Several successful object detectors have been proposed in recent years. A matter of concern to practitioners is which object detector and what configurations can achieve the best balance of accuracy and speed for a specific application. Methods that won the object detection challenges are usually optimized for accuracy but speed is also critical for applications in the real world [2]. Few methods, like YOLO, emphasize the need for speed at the cost of accuracy but they failed to demonstrate a full picture of the accuracy/speed trade-off except the effect of the input size.

A recent study by Google Research [2] investigated the speed/accuracy trade-off in an exhaustive and fair way for three object detection architectures: Faster R-CNN, R-FCN [24] and SSD. They implemented these algorithms using TensorFlow in a unified manner and compared the effect of different feature extractors, image size and number of proposals on speed and accuracy. They found that R-FCN and SSD are faster while Faster R-CNN is slower ($>$100ms) but more accurate. However, they did not cover YOLO which has outperformed SSD and R-FCN significantly in both accuracy and speed [8] and is the focus of this project. Besides, the factors that can affect the speed/accuracy trade-off covered in this project are different except resolution.

Chapter 3

Dataset

This chapter introduces the dataset we use in this project and presents some findings after doing data analysis that may have important implications for further experiment.

3.1 Berkeley DeepDrive Dataset

Large-scale image datasets such as ImageNet [25] and COCO [26] have promoted the recent advances in object detection methods. Currently, researchers tend to report the performance on COCO dataset to establish a unified comparison. However, these datasets contain common object categories that are not suitable for self-driving system. In this project, a dedicated dataset, ideally made from videos captured by high-res dash-cam, for self-driving applications is needed.

Berkeley DeepDrive (BDD) [11] is chosen for this project because it is a diverse and large scale dataset for computer vision tasks in the context of self-driving. This dataset contains 100k[1] images of which each is extracted from a 40-second long 720P high-res video at the 10th second and annotated with bounding boxes for objects in 10 categories: Bus, Traffic Light, Traffic Sign, Person, Bike, Truck, Motor, Car, Train, Rider. The resolution of all images is 1280×720. These images are split into 3 parts for training (70k images), validation (10k) and testing (20k) respectively. Since the test set is used for the Road Object Detection task in the 2018 Workshop on Autonomous Driving (WAD) challenge, its annotation is not publicly available and researchers need to submit the detection results on a specific website to obtain the evaluation score. This dataset covers 6 weather conditions (Clear, Partly Cloudy, Overcast, Rainy, Snowy, Foggy), 6 scene types (Residential, Highway, City Street, Parking Lot, Gas Stations,

[1] k means thousand.

Tunnel) and 3 different times of day (Dawn/Dusk, Daytime, Night).

Compared with BDD, other similar datasets such as Cityscapes [27] and KITTI [28] only provide limited diversity or scale. For instance, Cityscapes only has 5000 images.

3.2 Data Analysis

category	train	motor	rider	bike	bus	truck	person	light	sign	car
train	136	3002	4517	7210	11672	29971	91349	186117	239686	713211
val	15	452	649	1007	1597	4245	13262	26885	34908	102506

Table 3.1: The number of objects for different categories in training and validation set.

Data analysis is a preliminary step of almost all machine learning task. This section will present some quantitative analysis of the BDD dataset and introduce the training and validation set used for further experiment.

As shown in Table 3.1, the number of objects in different categories varies a lot. There are nearly one million cars but only more than one hundred trains in the dataset. This extreme class imbalance is likely to impede the object detector from recognizing objects of rare category such as *train*.

Figure 3.1 shows the distribution of object size in the entire training set and its breakdown for each category. According to COCO API, objects are divided into small objects ($h*w < 32^2$), large objects ($h*w > 96^2$) and medium objects in between. There are a large number of small objects and, therefore, the performance of detecting small objects is crucial to achieve high accuracy.

This dataset also provides image level tagging, including scene, weather condition and time of day. Table 3.2 shows the number of images in each tag. This kind of diversity especially the various times of day may have a negative effect on the detection accuracy.

It is worth noting that 137 images in the training set lack annotations probably because the annotation work has not been completed [2]yet. YOLO treats images without labels as negative samples but some of these images do contain objects. Therefore, it is necessary to exclude them from training set.

[2]Another evidence for this is that the dataset lacks annotations for instance segmentation though it was claimed to have provided.

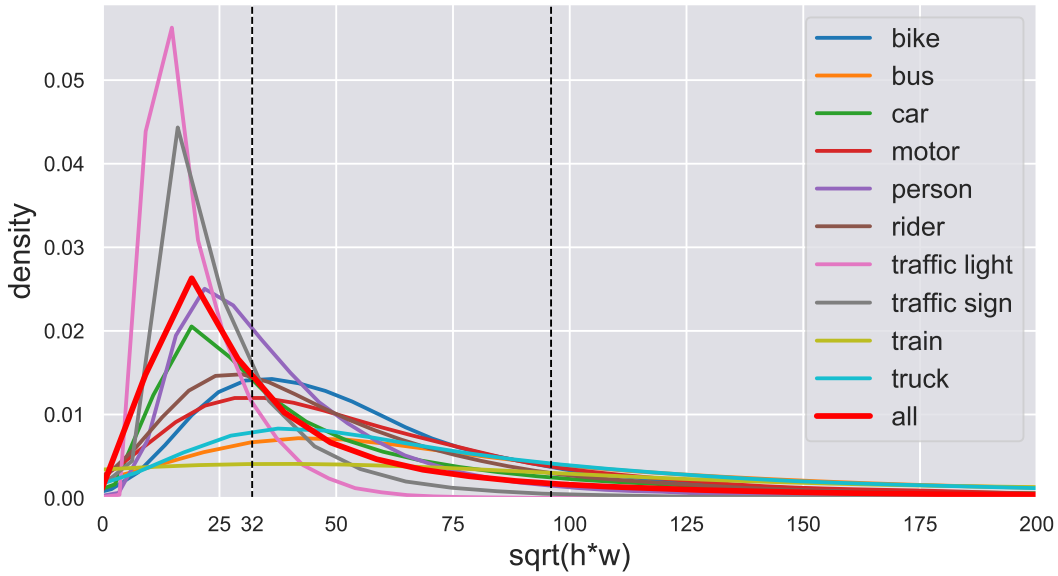


Figure 3.1: Distribution of object size in the training set for individual and all categories. The area under each density curve equals one. The majority of objects are very small.

Due to the limited time and GPU resource, all models except the final model which will be used to submit results for the WAD challenge are trained on a mini training set that contains 5k images randomly selected from the complete training set ($\sim$70k images). Since the annotation for test set is not publicly available and each account has limited submissions to the evaluation sever, half (i.e., 5k) of the validation set is held out as a local test set to report mAP in this project and the other half validation set is used to select the best weights during training.

Since the dataset contains many images that suffer from blurring or low contrast (e.g., at night), this project will exploit deblurring and contrast enhancement that might benefit object detection by mitigating these problems respectively.

To summarize, BDD is a very challenging object detection dataset for the following reasons. The most important reason is that the class imbalance as well as scarce training data for some categories (e.g., *train* and *motor*) results in extremely low mAP for these categories. Besides, there are a large number of small objects that are hard to recognize even for human. Furthermore, high-res image slows down the speed and makes it hard to detect small objects if the image is resized to a smaller resolution. What is more, the density of objects is higher because there are considerably more object instances per image than COCO (18.4 vs 7.3). Last but not least, the variation in scene, weather condition and time increases the complexity further.

weather	train	val
foggy	130	13
partly cloudy	4881	738
rainy	5070	738
snowy	5549	769
undefined	8119	1157
overcast	8770	1239
clear	37344	5346

(a)

scene	train	val
gas stations	27	7
tunnel	129	27
parking lot	377	49
undefined	361	53
residential	8074	1253
highway	17379	2499
city street	43516	6112

(b)

time of day	train	val
undefined	137	35
dawn/dusk	5027	778
night	27971	3929
daytime	36728	5258

(c)

Table 3.2: The number of images for different kinds of weather, scene and time of day in the training and validation set.

3.3 Class Imbalance

Class imbalance occurs when each class does not make up an equal proportion of the dataset. Extreme class imbalance is a crucial problem for classification tasks because it can lead to meaningless metrics or classifiers biased towards the majority class.

The "long-tail" effect in our dataset is a big impediment to the overall performance because mAP is calculated by averaging the mAP of each class, making class with extremely low mAP become a bottleneck. Data scarcity is also to blame for the poor performance on minority class like *train* because only few trains appear in the training set. The best but not the easiest way to resolve this issue is to add more data for these classes or use additional dataset. If this is not feasible, other approaches include cost-sensitive learning [29], transfer learning [30] and sampling [31]. Sampling is a simple strategy to address class imbalance either by replicating instances of the minority class or reducing instances of the majority class in order to create a more balanced dataset. We will adopt this approach to mitigate class imbalance in our final model.

Chapter 4

Algorithm

This chapter presents a detailed introduction of YOLO.

We select YOLO as our object detection system because it has a number of attractive features. One major advantage of YOLO is that it can achieve very good accuracy especially on mAP@0.5 while running very fast. Some two-stage detectors which can achieve higher accuracy, such as SNIPER [19], are subject to the speed constraint in self-driving system. Some one-stage detectors which can run at comparable speed, such as SSD [5], are not as accurate as YOLO. Another advantage is that YOLO is very popular and has been well studied and widely used, making it easier to learn and use than other detectors.

4.1 Networks

As illustrated in Figure 4.1, the architecture of YOLO is composed of feature extractor, detection network and yolo layer. This section will describe the first two components which form the bottom-up and top-down path in the feature pyramid network respectively.

4.1.1 Feature Extractor

YOLO is very fast and accurate partly because it uses an efficient backbone network called Darknet-53 since it has 53 convolutional layers. Darknet-53 can achieve equally good results as the state-of-the-art classifier ResNet-152 which has 152 layers on the 1000-class ImageNet classification task and, more importantly, is $2\times$ faster. As shown in Figure 4.1, Darknet-53 uses strided convolutional layer followed by al-

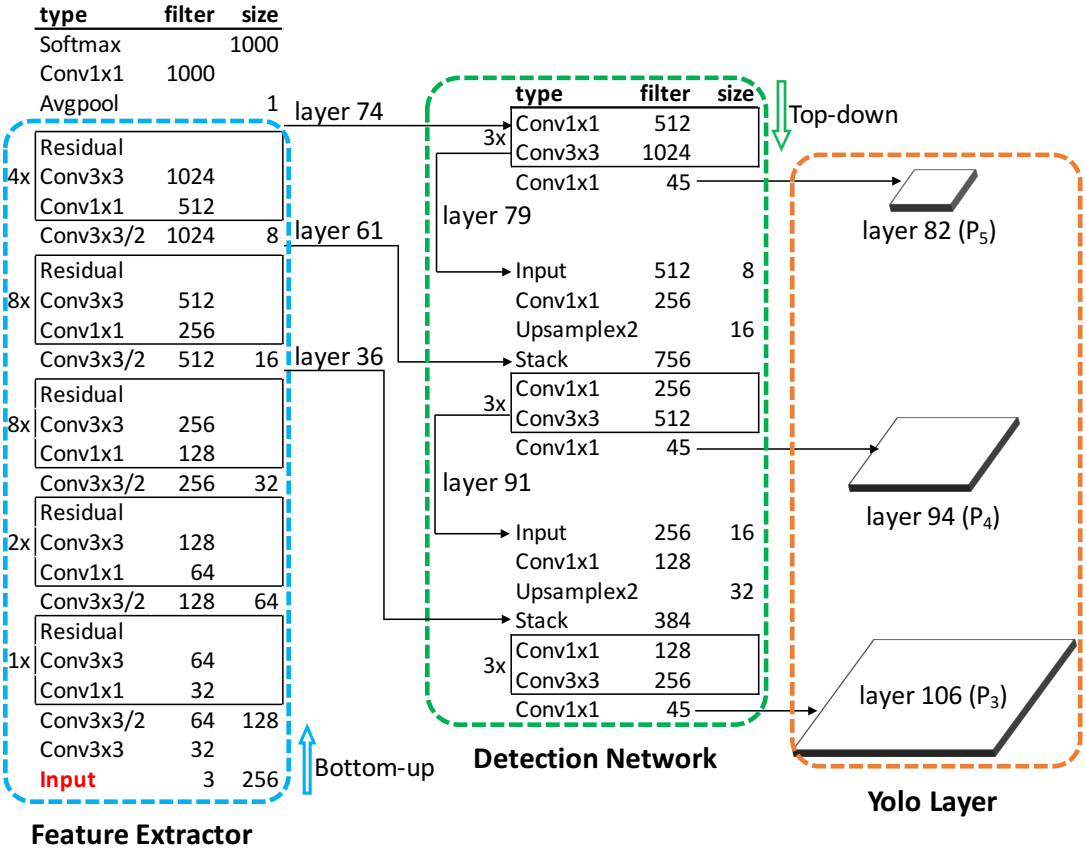


Figure 4.1: The architecture of YOLO consists of 3 main components: feature extractor, detection network and yolo layer. Conv3×3/2 denotes convolutional layer with kernel size 3×3 and stride 2. This figure assumes that the input image has width 256 and the total number of class is 10.

ternating 1×1 and 3×3 convolutional layers with residual connection. YOLO uses Darknet-53 without the last 3 layers as feature extractor because these 3 layers are only used for image classification.

As with pooling layers, convolutional layer with larger stride can be used to reduce the output dimension and increase context as well. Recent work [32] shows replacing max pooling layer with a stride-2 conv layer can get better results. This is why down-sampling is achieved by 3×3 stride-2 convolution instead of max or average pooling.

Residual block proposed by He et al. can solve the degradation problem when using extremely deep networks and ResNet-101 based on this building block has won several classification and detection challenges such as COCO detection. As illustrated in Figure 4.2, the output of the residual block is the sum of the output from the last convolutional layer and the input of the block. The skip connection (also called residual connection) acts as an identity function while the stacked layers in the block learn a residual function.

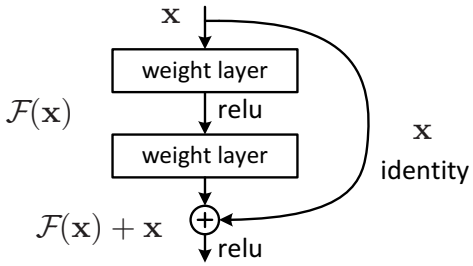


Figure 4.2: Residual block. Figure from [17].

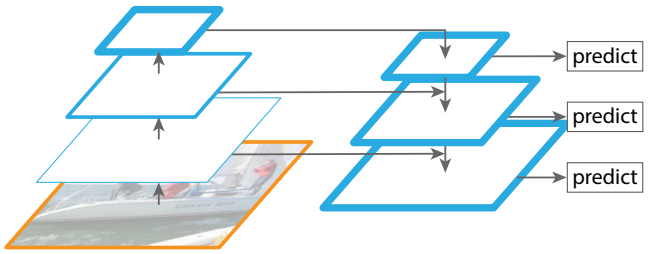


Figure 4.3: Feature Pyramid Network. Figure from [10].

4.1.2 Detection Network

In order to detect objects of different scales, detection network selects 3 feature maps at different levels from the feature extractor network. The last feature map (layer 74) of the feature extractor is followed by a group of successive 1×1 and 3×3 convolutional layers before performing detection on the yolo layer. Then, the feature map 3 layers before (layer 79) is upsampled by $2\times$ and merged with a larger feature map (layer 61) from the feature extractor using concatenation. This method keeps semantically strong features from high level feature map and finer-grained features from low level feature map. This merged feature map is also followed by a similar group of convolutional layers to generate prediction. The same design is performed again to utilize finer-grained features from a previous feature map (layer 36) which keeps sufficient details for detecting small objects.

In general, deep and high level pyramid layer has low resolution after downsampling and is responsible to detect large objects while shallow and low level pyramid layer is the opposite.

4.2 Yolo Layer

Yolo layer is a special layer that is responsible for generating bounding boxes and, if during training, computing loss based on the feature map produced by the detection network. This section will begin with anchor which plays an important role in yolo layers.

4.2.1 Anchor

The concept “anchor” was introduced by RPN [3] and now widely adopted by one-stage object detectors to handle dense objects. An anchor box, a.k.a bounding

box prior or default box, has a pre-defined size with respect to the network input size. Anchor boxes can be hand-crafted or computed from the ground truth boxes in the dataset using a clustering method such as K-Means.

YOLO generates a low-res feature map from the input image after running the feature extractor and detection network. Each cell in the feature map grid corresponds to a square area in the original image. A ground truth object is assigned to a cell where the center of its bounding box is located in. It is quite often the case that more than one objects are assigned to the same cell. YOLO proposes `#anchors` boxes at each cell and each predicted box is associated with an anchor box. With the help of anchors, multiple objects can be detected at each grid cell. An example is shown in Figure A.1.

Intuitively, it is hard to predict a big bounding box using a small anchor or vice versa. Therefore, the quality of the anchor boxes has a great impact on the detection accuracy. By default, YOLO uses 3 yolo layers and each layer has 3 anchors. These 9 anchors are chosen by running K-Means on COCO training set.

4.2.2 Prediction

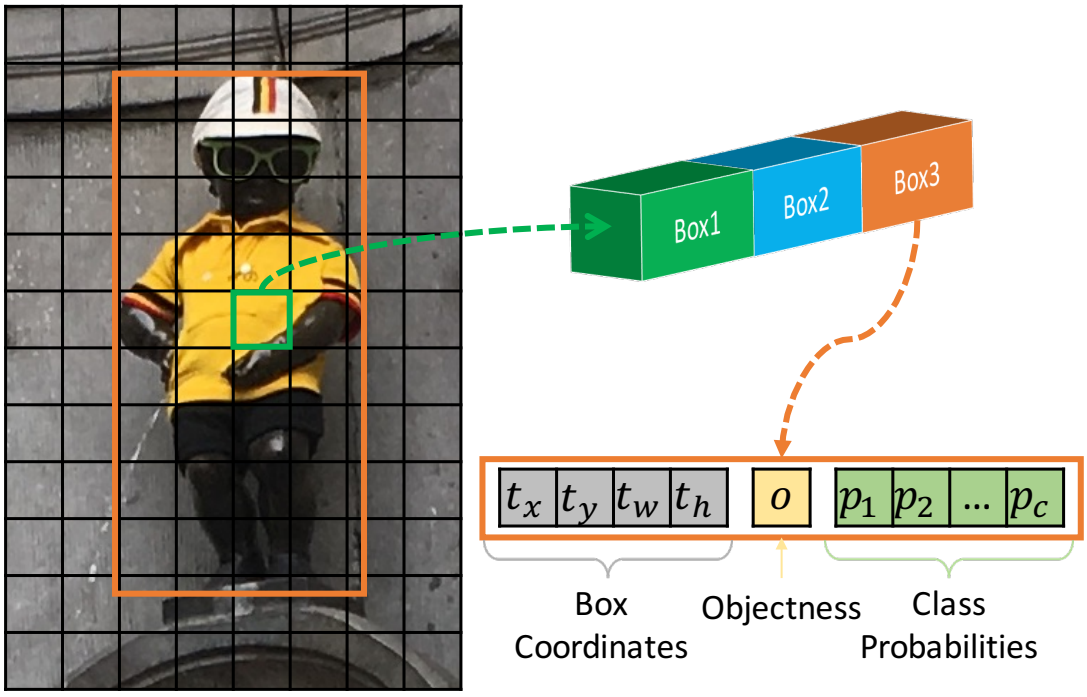


Figure 4.4: The input image is processed to generate a low-res feature map where each cell has 3 bounding box predictions each corresponding to an anchor and comprising 4 box coordinates, 1 objectness score and class probabilities.

The input of yolo layer is a 3D tensor which can be viewed as a grid encoding the

predicted bounding boxes along the depth dimension. At each grid cell, YOLO predicts 4 bounding box coordinates (t_x, t_y, t_w, t_h), 1 objectness score and 1 classification score associated with each class for every anchor assigned to the current yolo layer. A typical grid cell is shown in Figure 4.4. Therefore, the shape of the tensor is $height \times width \times [\#anchors \times (4+1+\#class)]$ and the total number of predicted bounding boxes in one yolo layer is $height \times width \times \#anchors$.

YOLO predicts the center coordinates as offsets in the grid cell and width (or height) relative to the anchor width (or height). As illustrated in Figure 4.5, if the cell has offset (c_x, c_y) from the top left corner of the grid and the anchor has shape (p_w, p_h), the predicted box is:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x & b_w &= p_w e^{t_w} \\ b_y &= \sigma(t_y) + c_y & b_h &= p_h e^{t_h} \end{aligned} \tag{4.1}$$

YOLO predicts the class of the object in the bounding box using multi-label classification (i.e., independent binary classification for each possible class) instead of multi-class classification because sometimes an object can have more than one labels (e.g., man and person). For each predicted bounding box, YOLO also predicts an objectness score which indicates whether the bounding box contains an object using logistic regression.

Location accuracy is a drawback of YOLO that prevents it from achieving high mAP at a strict IoU (e.g., 0.75) although it can obtain comparably good mAP@0.5. One possible reason for this result is the use of exponential function which is very sensitive to positive input to transform t_w and t_h to positives. An improvement we made is to replace exponential function with softplus $ln(1+e^x)$ which is more smooth. Figure 4.6 compares these two functions.

At test time, NMS (Non-Max Suppression) is applied to bounding box predictions from all yolo layers to generate the final detections. The algorithm of this post-processing step is shown in Algorithm 1. What NMS does is filtering out bounding boxes with low confidence which are probably corresponding to the background in the image and, more importantly, eliminating overlapping boxes. Each yolo layer outputs a bounding box prediction for each anchor at each grid cell. As a result, YOLO are likely to produce multiple detections for the same object and it is necessary to select the one with the highest confidence and discard others that are very close to it in terms of IoU but have lower confidence. This is why it is called non-maximum suppression.

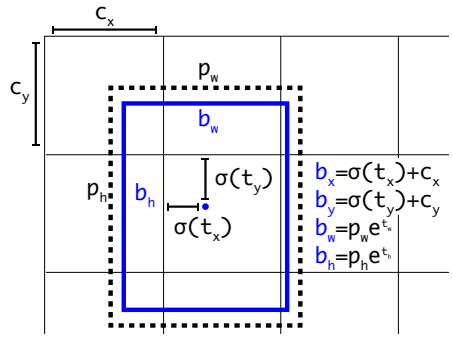


Figure 4.5: The bounding box is computed based on an anchor in a grid cell of the feature map. Specifically, the network predicts the center of the box (b_x, b_y) as offsets from the location of the grid cell (c_x, c_y) and the dimension (b_w, b_h) as some proportion of the anchor size (p_w, p_h). Figure from [8].

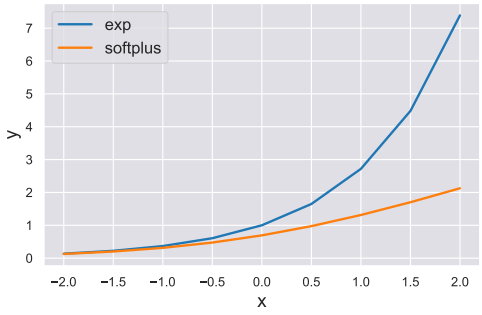


Figure 4.6: Exponential function versus Softplus. Exponential function is very steep while softplus is more smooth.

4.2.3 Loss Function

Each bounding box prediction consists of 4 coordinates (t_x, t_y, t_w, t_h), 1 objectness score o and, for each class c, a classification probability p_c. All values except t_w and t_h are squashed between 0 and 1 using sigmoid function. The loss function for the 4 coordinates is mean squared error while the loss for both objectness and classes is binary Cross Entropy.

Targets are defined as follows. Each ground truth object is assigned to one cell in the grid according to the location of its bounding box center. Furthermore, an object is assigned to only one anchor which has the highest overlap with the ground truth box, making each anchor specialize in predicting objects of certain size and aspect ratio. It means only one anchor in a specific cell on one of the 3 yolo layers is responsible for predicting this object. The target values for the prediction corresponding to this anchor are therefore clear. The objectness score $\hat{o}$ is 1 and the classification probability $\hat{p}_c$ is 1 for the class of the object and 0 for others. The ground truth coordinates $(\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h)$ can be computed by inverting equation 4.1.

If no ground truth object is assigned to a grid cell or an anchor, it only penalizes the objectness error of the corresponding prediction. An exception is that when this bounding box overlaps some ground truth box over a threshold (0.75 by default), the

loss of this prediction is ignored.

The loss function for a single prediction is shown in equation 4.2 where λ_{obj} and λ_{iou} are weights assigned to the loss of objectness and IoU respectively.

$$\begin{aligned} L = -\lambda_{obj}[\hat{o}\log o + (1-\hat{o})\log(1-o)] \\ -\hat{o}\Big\{ \sum_{c\in class} [\hat{p}_c \log p_c + (1-\hat{p}_c)\log(1-p_c)] \\ -\frac{1}{2}\lambda_{iou}\left[(t_x-\hat{t}_x)^2 + (t_y-\hat{t}_y)^2 + (t_w-\hat{t}_w)^2 + (t_h-\hat{t}_h)^2\right]\Big\} \end{aligned} \quad (4.2)$$

4.3 Data Augmentation

It is a common practice to augment the training data, such as cropping and horizontally flipping images, in order to increase the diversity of the dataset without collecting new data. Since a large training set is crucial to the performance of modern deep learning models, this strategy is very useful to increase the size of dataset when the data available is scarce. It is also useful to avoid overfitting due to irrelevant features and make the model robust to variations during testing even if a large dataset is available.

During training, YOLO applies horizontal flipping, random translation of up to 30% of the original image size and randomly adjusts hue, saturation and exposure in the HSV color space on the fly before feeding an image to the model. Examples of data augmentation are given in Figure A.2.

Besides image augmentation methods mentioned above, YOLO also uses multi-scale training method which benefits multi-scale object detection significantly. Instead of fixing the input size, it resizes the input image to a new size by a random factor between 0.667 and 1.5 every 10 batches. Since the model downsamples the image by a factor of 32, the new resolution is constrained to always be a multiple of 32. Multi-scale training can be seen as a kind of scaling augmentation strategy because it makes the same object appear in a different size when the corresponding image is used again during training. As a result, this method enables the model to work well for a variety of object sizes and also at different image resolutions.

Chapter 5

Experiments and Results

This chapter covers the experiments carried out. For each experiment, we will describe our motivation and experiment setup and then present the results followed by explanation and comparison to previous related work. Enough details such as important hyperparameter settings are given so that another researcher can reproduce these experiment results.

Experiments are done on an AWS p3.2xlarge instance which has a Tesla V100 GPU with 16 GB memory. Following the practice of recent object detectors [33], both YOLO and RetinaNet use backbones pretrained on the standard ImageNet-1k dataset which contains 1000 classes. During training, each model is evaluated on the first half of the validation set every 1000 batches and the model weights that perform the best[1] are chosen as the final weights to report mAP on the second half of the validation set. All mAPs are calculated by the COCO API with default parameters. The inference time is measured by averaging the time of processing 200 images on the Tesla V100 GPU and only includes the time of passing the image through the model[2] . The time of preprocessing such as loading and resizing images or post-processing such as NMS are excluded because it depends largely on the implementation and can be ignored if the code is well optimized.

5.1 Baseline

The first experiment aims to establish a baseline for comparison with the following experiments. The baseline model is obtained by training YOLO on the BDD mini

[1]This training schema has similar effect to early stopping which helps to avoid overfitting.

[2]It is worth mentioning that all timing information is measured when no other process is using the GPU because we notice it may take double time to process one image if the GPU has high load.

Experiment	Model	mAP	mAP_{50}	mAP_{75}	mAP_S	mAP_M	mAP_L	time (ms) [1]
Baseline	baseline	16.9	37.9	13.2	5.3	21.3	34.8	11.4±0.2
Scales	2 scales	12.9	29.7	10.0	2.1	16.9	33.1	9.7±1.3
	5 scales	18.5	39.6	14.9	8.3	21.6	32.6	14.5±0.2
Anchors	2 anchors	16.7	38.1	12.5	5.3	21.3	33.0	11.4±0.6
	5 anchors	17.1	38.0	13.2	5.7	21.3	34.6	11.8±0.2
Resolution [2]	baseline_512	14.3	32.3	11.1	3.6	18.5	34.3	8.6±0.2
	baseline_768	18.0	40.1	13.9	6.4	22.6	32.5	14.5±0.2
	baseline_896	18.5	41.2	14.4	7.6	23.1	31.3	17.0±1.5
	baseline_1024	18.5	41.8	14.0	8.7	23.0	28.7	20.4±1.3
	baseline_1280	16.9	39.0	12.5	8.7	21.4	22.7	29.9±1.5
	896	19.7	42.6	15.8	8.4	24.4	36.3	17.3±0.4
	1024	20.4	44.2	16.5	8.2	25.0	36.3	20.8±1.3
	1280	19.5	41.2	16.0	9.2	24.2	33.2	30.7±2.1
Enhancement [3]	baseline_blur	14.9	33.5	11.3	5.6	21.0	38.2	—
	baseline_sharp	17.0	37.4	13.5	7.0	23.8	44.3	—
	baseline_night	15.9	38.7	10.9	6.8	21.8	29.0	—
	baseline_ce	15.3	37.8	9.9	6.3	20.7	27.6	—
RetinaNet	RetinaNet	18.7	37.5	16.2	5.0	23.7	38.6	54.0±1.9
Final	final	28.0	55.4	25.0	13.6	32.1	47.3	23.2±0.4

[1] Inference time is present as mean ± confidence_interval at 95% confidence level. Confidence interval takes both standard error and sample size into account.

[2] baseline_{512,768,896,1024,1280} means the baseline model with input size 512×288, 768×448, 896×512, 1024×576, 1280×736 respectively.

[3] These models use different test sets from others.

Table 5.1: Experiment results. Although mAP@.5 and mAP@.75 are provided for completeness, the mAP throughout refers to mAP@[.5:0.95] specifically because it is the main metric used by COCO. In addition, this table also shows the breakdown of mAP in small, medium and large objects.

training set introduced in 3.2.

There are a number of implementations for YOLO in addition to the official Darknet provided by the author of YOLO. However, most of them struggle reproducing the same results after training using the same data and configuration. Instead of using the

official implementation of YOLO, we adopt AlexeyAB's Darknet[3] which is forked from the official repository because this fork has been optimized in several ways and offers more features than the official code while achieving equal or better performance. For example, one facility provided by this fork is that during training mAP can be calculated on the validation set periodically and the interim model with the highest mAP so far will be saved.

Important hyperparameter settings and justification are given below. All YOLO models in the following experiments use these settings unless otherwise stated.

We set `width=640` and `height=384` respectively. YOLO and many other object detection systems will resize the input image before feeding into the network to a specific size which is configurable. We did not use the raw resolution (1280×720) for time and memory reason. Moreover, YOLO requires the input dimension to be divisible by 32 because the input to the network will be downsampled with stride 32. This is why the height is 384 instead of 360. YOLO will also keep the original aspect ratio by padding which means there are 12 pixels of padding near the top and bottom.

We use a batch size of 32 instead of the default 64 which is tuned for COCO dataset because there are much more objects per image in the BDD dataset.

We set `max_batches=16000` for training around 100 epochs on 5000 images to ensure that the performance will eventually saturate.

We set `steps=12800,14400` so the current learning rate will decay by 0.1 after 80% and 90% of the entire training process respectively. This learning rate schedule is a common optimization procedure that can reduce training time and increase performance by learning quickly earlier and fine tuning later.

We set `filters=45` for the layer prior to each yolo layer because the number of output filters is supposed to be `[(#classes + 5) * #anchors]` where 5 means 4 bounding box coordinates plus 1 objectness prediction.

We follow the convention of YOLO and generate 9 anchors by clustering object dimensions in the training set using K-Means with a customized distance function that calculates the IoU instead of Euclidean distance. The visualization of clustering results and anchors are illustrated in Figure 5.1. The anchor dimensions in the configuration file need to be normalized with respect to the image resolution after resizing.

We set `random=1` to enable multi-scale training which augments training data by rescaling the input size by a random factor between $\frac{2}{3}$ and $\frac{3}{2}$ every 10 batches.

From the training curve in Figure 5.2, we can see that the mAP plateaus after about

[3] https://github.com/AlexeyAB/darknet

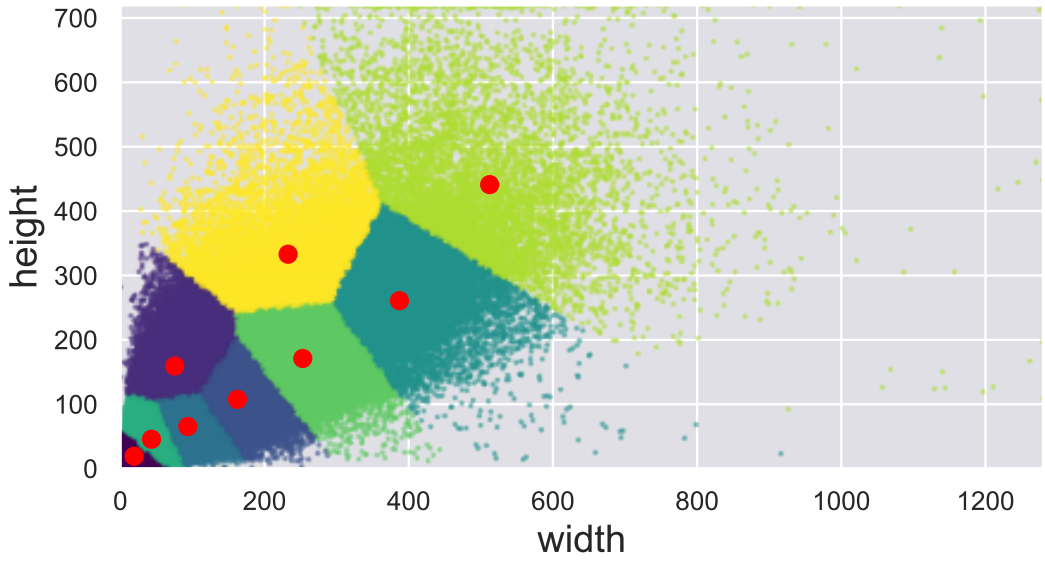


Figure 5.1: Clustering of the ground truth bounding boxes in BDD training set using K-Means. Red centroids represent anchors.

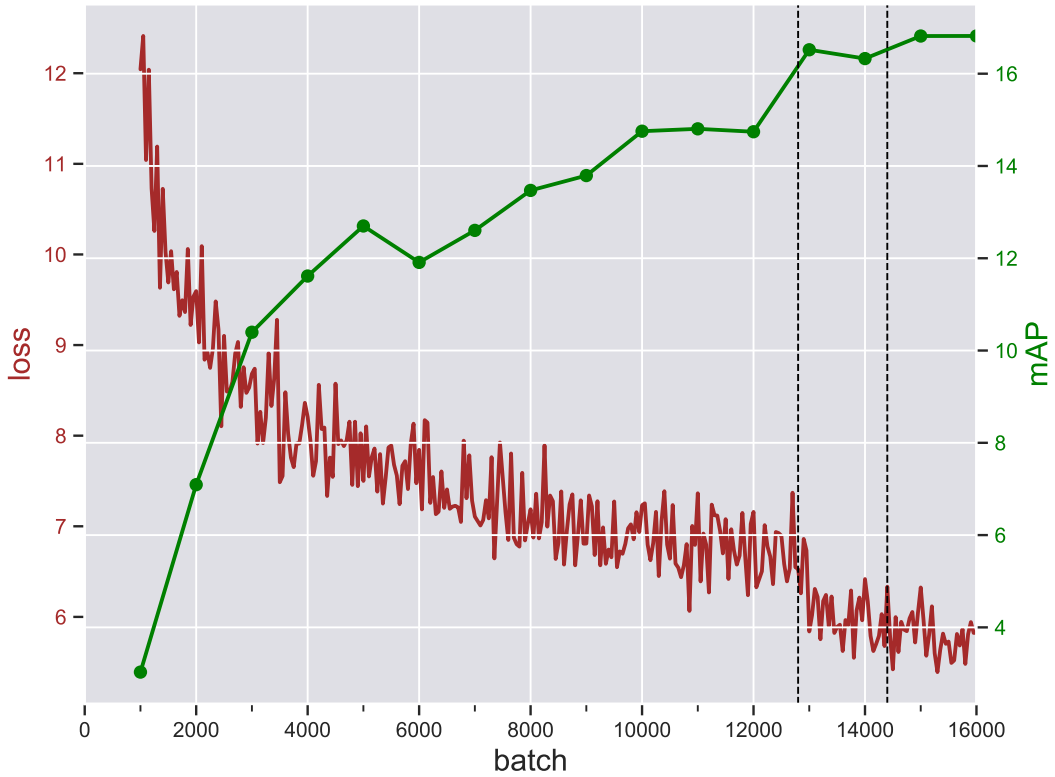


Figure 5.2: Loss and validation mAP of Baseline model during training.

10^4 batches but increases (equivalently, the loss decreases) again after the first learning rate schedule point at 12800 batches.

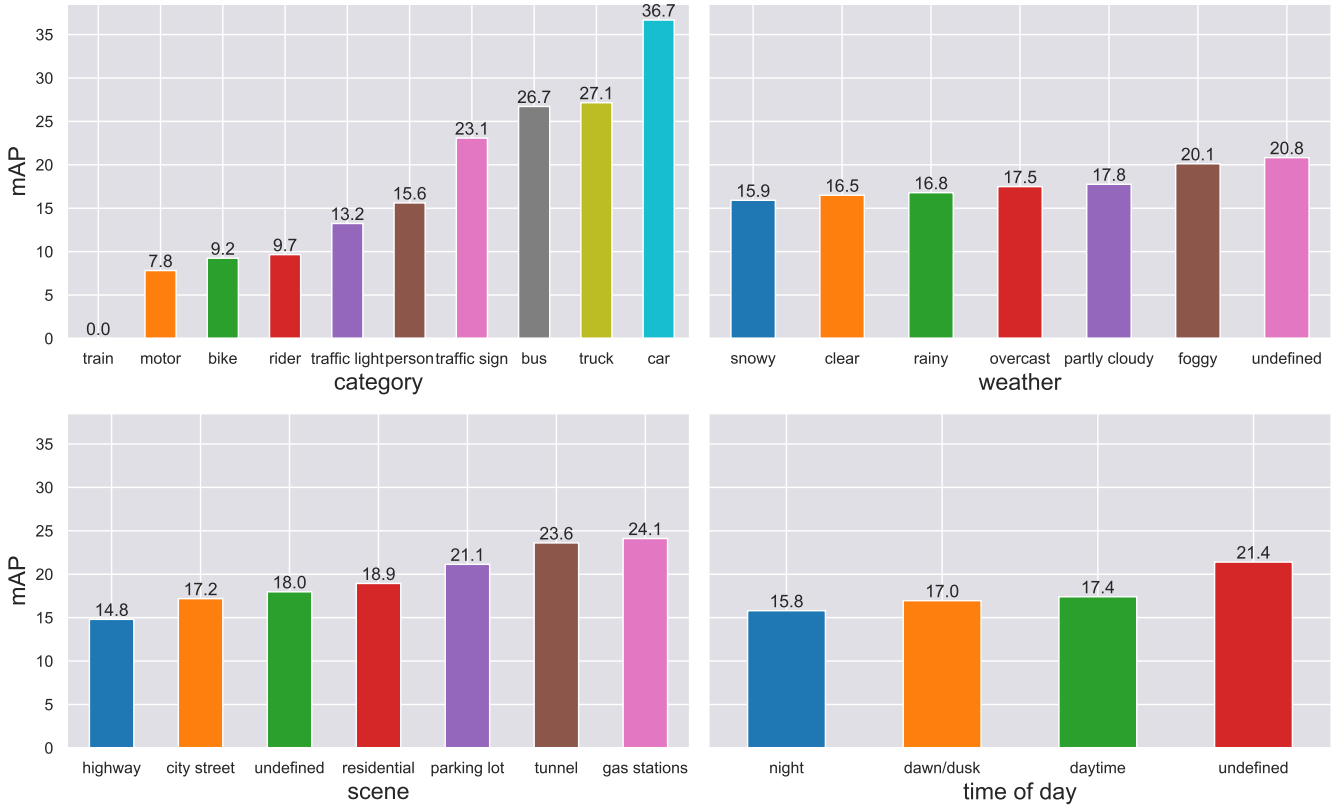


Figure 5.3: The breakdown of mAP in different categories and image tags, namely weather, scene and time of day for the baseline model.

The results of this experiment as well as all experiments below are shown in Table 5.1. We notice that the object size has significant implications for mAP. Compared with large objects, the mAP for small objects is quite low and this cumbers the overall mAP since small objects account for a large proportion in the dataset.

As shown in Figure 5.3, the mAP is further divided by object category and scene, weather condition, time of day in the image to facilitate analysis of the performance.

The mAPs vary a lot across different categories. For example, the best mAP reaches 36.7 for *car* while the worst mAP is 0 for *train*. The huge gap is due to the data scarcity and class imbalance in the training set. For instance, *train* only appears 136 times in the complete training set, making its mAP extremely low and become a bottleneck of the overall performance. We also notice that the mAP for *traffic lights* is very low despite the fact that there are a large number of such objects in the dataset. One important reason is that most traffic lights, as can be seen from Figure 3.1, are small objects and hence hard to detect.

In contrast, the mAPs for different weather conditions are rather uniform. It is

in line with our intuition that *snowy* days make it relatively hard to perform detection since objects may be covered by snow and less color information is available. However, good weather condition does not imply good performance since the mAPs for *overcast* and *rainy* days are slightly better than that of *clear* days. Higher mAP in the weather *foggy* and *undefined* can be attributed to the small sample size. We find that the hard class *train* does not appear in these two weather conditions in the validation set and the calculation of mAP in COCO API will ignore this class completely. Besides, weather *foggy* only has a very small number of images in the validation set and might lead to inaccurate mAP calculation.

Similar results are also observed for different times of the day with the exception of *undefined* which is caused by the absence of hard class *train* and lack of sufficient samples in the validation set. Low light condition especially at night does have a negative impact on the performance of object detection.

The mAPs in different scenes seem to be quite different but this can also be explained by the small sample size and hard class. For instance, *train* only appears in *city street* and *highway* and, furthermore, most appear in *highway*, making the mAP for *highway* relatively low.

5.2 Detection Scales

YOLO detects objects at 3 scales using feature pyramid levels from P_5 to P_3, where P_l is downsampled by a factor of 2^l. During downsampling, the context of too small objects might be lost.

In this experiment, we extend the architecture to use high resolution feature pyramid levels and detect objects at 5 scales from P_5 to P_1. Briefly speaking, P_5 is the highest level in the feature pyramid. After being upsampled by a factor of 2, P_5 is concatenated with a previous feature map which has equal size to produce P_4. P_3 is produced from P_4 in a similar way. We replicate this structure and append another 2 detection layers on P_2 and P_1. This model is expected to perform better for small objects but also be slower.

As a comparison, we did another experiment by removing the last level P_3 and only use 2 scales. It does not make sense to remove only P_5 or P_4 because this has little help for reducing computation.

These architecture changes also require to modify the configuration for anchors and we use K-Means to generate 3 anchors at each scale as before.

From the experiment results in Table 5.1, it conforms to our expectation that the 5-scale model can improve the mAP significantly especially for small objects though it is also slower while using only 2 scales leads to disaster for both small and medium objects. It is, however, unexpected to observe the degradation of performance for large objects when using 5 scales and this might result from the bad choice of anchors. When 15 anchors are generated using K-Means and distributed evenly across 5 scales, we notice that most objects, including many large objects, end up being assigned to the lower level feature maps during training, making higher level feature maps ineffective and impeding the performance for large objects.

5.3 Anchor Density

Anchor is one of most important design components in an one-stage detector. It was shown that RetinaNet performs the best when using 6 anchors of 2 scales and 3 aspect ratios per detection level [6] but it is not known how YOLO performs while varying the number of anchors. Intuitively, anchors that have a good coverage of the space of possible object boxes will yield better results. Motivated by this hypothesis, this experiment will study the effect of anchor density on the performance of YOLO.

The baseline model follows the default configuration in YOLO which uses 9 anchors and divides them evenly across 3 scales. In this experiment, we have trained another 2 models using 2 and 5 anchors at each scale respectively. These anchors are generated by the same clustering algorithm used in 5.1. As can be seen from Table 5.1, only slight improvement is seen when increasing the number of anchors from 2 to 5 and one possible reason for this is that using 2 anchors per scale already covers a wide variety of objects in the dataset. To our surprise, the time incurred by more anchors is negligible probably because the CNN dominates the computation complexity.

Lin et al. investigates the effect of anchor density for RetinaNet and found that using 9 anchors per scale can improve the mAP by 4 points compared to just using a single square anchor but the performance saturates when increasing the number of anchors further. However, we could not observe significant improvement in our experiment when increasing the number of anchors probably because RetinaNet and YOLO use anchors that are generated in different ways. RetinaNet uses dataset agnostic anchors by tiling a collection of boxes at predefined aspect ratios (e.g., $[0.5, 1, 2]$) and scales ($\text{SCALE} * 2^{k/4}, \quad 0 < k \leq 3$). In contrast, we use anchors generated by clustering the ground truth boxes in the dataset and 6 anchors in total may already match

most objects well. Having said that, we only test 2,3,5 anchors per scale and maybe a wider range of parameters are needed in order to show the difference.

5.4 Image Resolution

Previous studies have shown that high resolution can significantly improve the detection accuracy especially for small objects. Both YOLO and RetinaNet have reported the accuracy/speed trade-off on COCO test-dev while varying the image resolution. This experiment will try to reproduce the results of YOLO on a new dataset and further study the effect of resolution. We also plot the accuracy/speed trade-off curve for this experiment in Figure 5.4.

Firstly, we test the baseline model which was trained at 640×384 using various resolutions. YOLO is a fully convolutional network which allows to process images of arbitrary size[4] during inference regardless of the size used for training. Furthermore, this model is trained with multi-scale enabled and hence can work well at a range of input resolutions.

As shown in Table 5.1 and Figure 5.4, the results conform to previous findings that higher resolution yields better mAP but also makes detection slower. In addition, we also observe that the mAP decreases when the input size is larger than 1024×576 because increasing input resolution helps to improve accuracy for small objects, but in the meantime, lowers accuracy for large objects. It is easy to understand the first case because high-res images allow more small objects to be resolved. However, it is not straightforward to explain why high resolution leads to worse results for large objects. One reason for this observation is that the detector only works well for objects in a range of sizes that appear in training data and does not work well for too large objects beyond that range. In general, YOLO performs well for small objects in high resolution input and large objects in low resolution input.

Then, a few models are trained and tested using different resolutions, including 896×512, 1024×576 and 1280×736. These models inherit configurations from the baseline model except that width, height and anchor size are changed accordingly.

Compared with the baseline model using a higher resolution for inference, the model trained at the same high resolution can achieve much better results without any compromise on speed. This difference can also be explained by the scale variation as above. The baseline model is trained using image size of 640×384 and, as a result,

[4] As long as it is a multiple of 32.

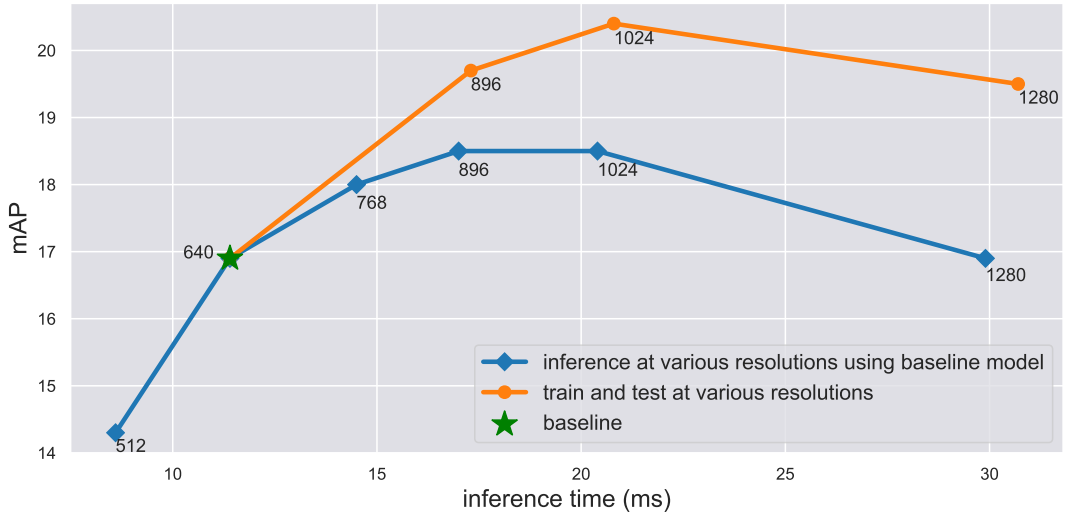


Figure 5.4: Accuracy/speed trade-off while varying resolution

large objects in high-res images are larger than objects that are used for training and thus are hard to detect (i.e., hard examples). In contrast, models in this experiment are trained and tested at a higher resolution and can perform well on these large objects.

Surprisingly, we also noticed that the performance saturates and declines when increasing the resolution further. This also accords with earlier observations from Table 1(e) in [6] where the performance of RetinaNet for large objects drops slightly when the resolution is higher than 600. It is difficult to explain this result but it is very likely to be related to context modeling. Downsample does not only reduce computation but also helps to broaden context. Higher downsample ratio means larger context. For instance, an image which is downsampled by a factor of 8 means every 8 pixels along the width or height dimension will be reduced to 1 pixel. Multiple convolutional layers stacked together also help to increase context but are less effective than downsample methods. YOLO downsamples the input image by a factor of 32 which means large objects (i.e., $area > 96 \times 96$) will occupy at least 3×3 area in the highest feature map. YOLO detects objects at each location which is supposed to have included all contextual information, like edge, about the objects assigned to this location. As a consequence, it is difficult to detect very large objects. One possible solution is to use deeper feature extractor that downsamples the feature map further.

5.5 Image Enhancement

Image enhancement is the procedure of improving the quality of original images so that the results are better for displaying or further processing [34]. Common methods

of image enhancement include deblurring and the adjustment of color balance, contrast and sharpness. This section will investigate the effect of two approaches, deblurring and contrast enhancement, on object detection. These image preprocessing techniques can enhance the image quality for human eyes and we expect them to improve the detection accuracy during inference time as well.

5.5.1 Deblurring

Images taken by a shaking camera on the self-driving car are likely to be blurred, making objects indistinct and fuzzy. Image blurring in our dataset results from not only camera shake but also object motion because other vehicles are also moving at different speeds, making it very hard to restore the sharp image.

To evaluate the effect of deblurring, we intended to compare the detection accuracy of YOLO on blurred images and sharp images restored after removing blurs. However, the difficulty is there may not exist a deblurring tool that works well for our project. Deblurring is a very challenging task because image blurring can be caused by various sources, including object motion, camera shake and scene depth variation. Since conventional deblurring methods such as Wiener filter assume the blur kernel is partially known, they can only handle a specific type of blurs such as camera motion and does not perform well for non-uniform blurs [35]. DeblurGAN [36] is a recent machine learning based deblurring method that is intended to handle these complicated blurs caused by various sources in an intelligent mechanism. However, there are a few reasons that stop us from adopting this approach. First of all, DeblurGAN is a learned method that requires to be trained on our own dataset to work well. It is shown that the best results are obtained after training on a combination of synthetically generated blurred images and blurred images taken from the real world. The lack of such data makes it difficult to train a DeblurGAN model and, additionally, the training phase takes 6 days. What is even worse is that DeblurGAN takes 0.85s to process one image, making it useless for our task which is constrained within 0.1s.

Due to the above reasons, we decide to give up deblurring and instead compare the detection accuracy on original sharp images and synthetically blurred images generated by applying random motion blur on the sharp ones. As no previous study has shown to what degree deblurring can affect the performance, this can tell us exactly how much the accuracy can be improved if a deblurring oracle exists and all blurred images can be 100% restored.

Using the baseline model trained in 5.1, we test on 1000 images chosen from the validation set before and after applying random motion blur whose kernel size ranges from 3 to 10 and angle ranges from 0 to 360. Figure 5.5a shows an image after blurring. As shown in Table 5.1, the detection accuracy degrades 2.1 points if motion blurs appear in the image, which also indicates the upper limit of improvement in accuracy if deblurring is employed.

This conforms to our expectation that blurred images result in poor performance and deblurring benefits object detection. Since most images in the whole dataset are not blurred, the improvement for the overall performance should be very limited and this enhancement is thus not worthwhile for our task.

It is worth mentioning that Kupyn et al. also tests the performance of YOLO after applying DeblurGAN to blurred images and a higher recall is obtained after deblurring. However, they use synthetically generated blurred images that stimulate camera shake instead of real blurred images which tend to be more difficult to recover. Their experiment also saw a much lower precision, meaning the effectiveness of DeblurGAN is uncertain if measured under the metric mAP which they did not report.

5.5.2 Contrast Enhancement

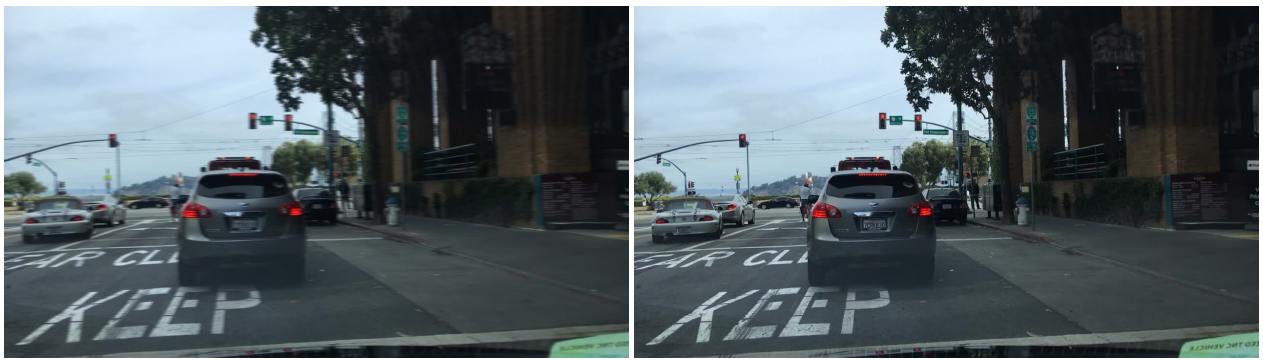
A large number of images in BDD are captured at night. Moreover, self-driving cars are expected to work under low-illumination conditions as well. Images at night often suffer from low contrast and, as can be seen from Figure 5.3, result in worse performance. Contrast enhancement (CE) can help to adjusts the contrast in over- or underexposed images and provide more distinctive features for object detectors.

In order to evaluate the effect of contrast enhancement for object detection, we measure the performance of the baseline model on a tiny validation set containing 1000 images captured at night before and after contrast enhancement respectively.

Ying et al. propose a new contrast enhancement algorithm that can automatically improve the contrast based on exposure fusion and achieve the state-of-the-art performance. We use the python implementation[5] of this algorithm and an example after enhancing the contrast is shown in Figure 5.6. It is meaningless to compare the inference time including the time of enhancement because the CE tool mentioned above is not optimized and takes too long (about 5s) to process one image.

As shown in Table 5.1, to our surprise, the mAP drops slightly after contrast en-

[5]https://github.com/AndyHuang1995/Image-Contrast-Enhancement



(a) blurred (b) original sharp image

Figure 5.5: A synthetically blurred image and its sharp counterpart which represents the target of deblurring.

(a) original image (b) after contrast enhancement

Figure 5.6: An example of contrast enhancement.

hancement. While the use of contrast enhancement to improve the performance of object detection has not been investigated before, it is a surprising finding that CE is not helpful for object detectors although it produces better visual effects for humans. One possible reason is that the object detector has learned some sort of feature extractor for the natural low-light scene and fails to generalize well to the synthetic scene after contrast enhancement.

5.6 Comparison with RetinaNet

This experiment aims to compare the performance of YOLO and RetinaNet on BDD dataset. Both RetinaNet and YOLO are one-stage detectors and they share many similarities in the design, such as feature pyramid and anchor. YOLO is able to run much faster but fails to achieve the same accuracy as RetinaNet in terms of COCO

mAP. This experiment will try to figure out whether RetinaNet can achieve much higher accuracy than YOLO while still meeting the speed constraint of the self-driving system.

We adopt the official implementation of RetinaNet in Detectron[6] provided by Facebook Research for this experiment. Our configuration is based on RetinaNet-101-FPN-800 which was used to obtain the published results on COCO with the following notable configurations. We use scales ranging from 640 to 800 with a step size of 32 for training and scale 720 for testing. The scale here means the shorter side of the input image after being resized while keeping the original aspect ratio. For instance, the resolution for testing is 1280×720. The reason why we do not keep resolution consistent with the baseline model for fair comparison is that the original RetinaNet was exactly trained using these resolutions and we tried using the same resolution as the baseline model, for example, but got much worse results. Other configurations such as the learning rate, schedule point, the number of GPU, classes, iterations are also modified accordingly.

From Table 5.1, we can see that RetinaNet outperforms the baseline model of YOLO by 1.8 points but it should be pointed out that RetinaNet uses a much higher resolution for both training and inference than YOLO (1280×720 versus 640×384). Experiment results in 5.4 show YOLO can achieve higher accuracy if it uses the same resolution. Since YOLO also runs much faster, RetinaNet shows no advantage over YOLO in our task.

It is reported that RetinaNet with ResNet-101 and FPN can yield mAP nearly 6 points higher than that of YOLO on COCO dataset (39.1 vs 33.0). A possible explanation for the discrepancy between our result and the published one is that our hyperparameters and hardware might prevent RetinaNet from yielding better results. For example, the original RetinaNet was trained using 8 GPUs with data parallel synchronous SGD which enables it to benefit from large batch size (e.g., 16). Since only one GPU is available for this project, only a small batch size of 2 is allowed in our experiment due to the memory restriction. Our extremely small batch size causes the learning to be very noisy and also hinders effective batch normalization which usually benefits from large batch size [38]. In contrast, YOLO handles the memory limit in a smart way: it divides a batch (e.g., 32 images) into several mini-batches (e.g., 2 images) and only runs a mini-batch on a GPU at a time. The weights are updated once all the loss from these mini-batches are collected.

[6] https://github.com/facebookresearch/Detectron

This experiment has important implications for selecting the right object detection method for a specific application. As pointed out by Huang et al., it is not easy to perform apples-to-apples comparison between different object detection methods because they differ in many ways such as input image resolution, detection scales, choice of anchors and deep learning framework. Therefore, those published results should be interpreted with caution.

Firstly, it is unwise to only compare the mAP which usually does not reveal the full picture. RetinaNet is reported to perform much better than YOLO on COCO dataset. One of the most important reasons is that RetinaNet resizes the input image such that the shorter dimension is 800 while keeping the aspect ratio unchanged whereas YOLO makes the longer dimension be 608. Obviously, the input of RetinaNet has a much larger resolution and preserves more information. Another reason is that RetinaNet detects objects on pyramid levels from P_3 to P_7[7] of which each uses 9 anchors of 3 scales and 3 aspect ratios whereas YOLO detects objects at 3 scales using 3 anchors at each scale. It is easy to see that YOLO trades accuracy for speed while RetinaNet improves the accuracy by sacrificing speed.

Secondly, the speed reported in previous papers might not be comparable even if they are measured on the same hardware platform. Speed is usually reported without giving many details about how it was measured. After checking the code of both YOLO and RetinaNet, we found that YOLO does not include the time of either preprocessing (e.g., resize) or post-processing (e.g., NMS) while RetinaNet does. Besides, speed depends on the implementation to a large extent and bad implementation can affect the speed significantly. For instance, the hand-written `resize` function in YOLO takes about 20ms to process one image while the `cvResize` function provided by OpenCV only takes less than 1ms.

5.7 Final Model

We aim to train the final model that can achieve high accuracy while still meeting speed requirements based on the knowledge we have learnt through previous experiments. We will describe how this model combines a number of optimizations and evolves from the baseline to nearly the state-of-the-art below.

Following the practice in [16, 39, 21], we conduct ablation studies to evaluate the contribution of different optimization techniques. Table 5.2 presents a summary of

[7] The resolution of P_l is 2^l times smaller than the input image

	baseline	interim						final
complete training set		√	√	√	√	√	√	√
balanced training set			√	√	√	√	√	√
exp → softplus				√	√	√	√	√
5 scales					√	√	√	√
new anchors						√	√	√
resolution 896×512							√	√
training 2× longer								√
mAP	16.9	19.4	21.8	22.5	24.1	24.6	26.4	28.0

Table 5.2: Ablation studies of the final model. The baseline model is combined with several optimizations to form the final model.

the ablation experiment results. It should be pointed out that all interim models are trained on the complete training set for 20000 batches to save time ($\sim$ 9 epochs) while the final model is trained 2× longer (i.e., 40000 batches). Learning rate schedule is modified accordingly. The mAP in the table above is reported on the same validation set as before because we can only submit results for test set to the evaluation sever for a limited number of times.

Compared with the baseline model, this model benefits a lot from the diversity of the complete training set which is more than 10 times larger than the mini training set used for previous experiments.

Furthermore, the training set is balanced by sampling mentioned in 3.3 such that an image that contains an object of a rare class will be used more often for training. Classes and their corresponding weights which are selected manually are shown in Table A.1. The weight of an image is the maximum among the weights associated with all the classes it contains. For instance, an image that contains a motor and a car will be seen 50 times more often than an image that only contains a car. Using this simple strategy to balance the dataset improves the mAP by 2.5 points, showing the importance of dealing with class imbalance.

As previously discussed in 4.2.3, we replace exponential function with softplus to improve the bounding boxes prediction. This minor optimization helps to increase the mAP by 0.7 point.

Next, we append another 2 yolo layers at scale P_2 and P_1 respectively as we did in 5.2 to improve the performance of small object detection. This gives us a significant increase in mAP.

Unlike previous experiment, we use 9 hand-crafted anchors because we learned from 5.2 that anchors generated by clustering may impede the effectiveness of low-res scales. The difference between the anchors used in the baseline and the new anchors here are highlighted in Table A.2. These new anchors result in more even distribution in different scales. A slight improvement on the mAP is seen though we expect more.

Following the findings in 5.4, this model is then trained and tested at a higher resolution to increase the mAP further by 1.8 points. We use 896×512 instead of 1024×576 for both speed and memory reason.

Lastly, the mAP rises to 28 after training 2× longer. It is worth mentioning that it takes about 8 days to train the final model with CUDNN enabled. Some output examples can be seen in Figure B.2.

Finally, we submitted the detection results for the test set to the evaluation sever which measures mAP@0.75. Our final model gains 25.75 mAP@0.75 on the BDD test set, surpassing the third place (20.66) by 5 points on the leaderboard of the Road Object Detection Challenge for CVPR 2018 Workshop on Autonomous Driving. Despite the impressive accuracy, our model can also run at 43 FPS on a Tesla V100 GPU. To the best of our knowledge, our final model is the fastest that can achieve similar or better accuracy on BDD test set.

We notice that there is still a large gap in mAP@0.75 compared to the first (33.10) and second place (29.69) on the leaderboard of this challenge. The second place uses an one-stage detector called CFENet [39] which is based on SSD. CFENet inherits the architecture of SSD and introduces a Comprehensive Feature Enhancement module which enhances the shallow features of SSD for detecting small objects. This method outperforms SSD significantly and is slightly better than YOLO on COCO dataset. On BDD test set, the single-scale version of this method with input size of 800×800 can achieve 22.34 mAP@0.75 and 21 FPS while the multi-scale version can achieve 29.69 mAP@0.75. However, it is not clear what multiple inference strategy they adopted to boost mAP about 7 points and how fast this multi-scale version is. Compared to their single scale model, our approach using existing method is able to achieve better mAP at a resolution of 896×512 which means less input data in terms of the number of pixels. Since there is no literature about the method used by the first place, it is possible that they rely on ensemble or multiple inference that are too slow for practical usage. For instance, an ensemble using NMS to merge the results of 10 models each trained for a distinct class separately might address the class imbalance problem and improve the overall performance significantly.

Chapter 6

Conclusion

The purpose of this chapter is to summarize our findings and then discuss a few limitations of our approach as well as ideas for future work.

6.1 Findings

This research set out with the aim of understanding the trade-off between accuracy and speed of modern object detectors when applied to self-driving cars. In spite of being limited by the GPU resource, a number of experiments were undertaken to evaluate the effect of different ways on the accuracy/speed trade-off using a state-of-the-art object detector YOLO and a diverse driving dataset BDD.

Two different aspects, namely model and data, were explored to trade-off between accuracy and speed.

Firstly, we explored possible ways to affect this trade-off by modifying the model configurations especially the two key design factors: detection scale and anchor. The effect of pruning or adding detection scales was examined and additional detection scales using fine-grained features from high-res feature maps benefit the detection accuracy significantly especially for small objects. In contrast, it is not very encouraging to find that the number of anchors per scale has little impact on the performance.

Secondly, we investigated how image resolution and different image enhancement techniques can affect the accuracy and speed of object detection. Resolution is among the easiest and also the most effective ways to trade-off accuracy vs speed. This research supports evidence from previous studies that, in general, high-res images benefit detection accuracy especially for small objects but also found that there exists an appropriate resolution such that higher or lower resolution will hinder the detection of

large or small objects respectively and thus results in worse performance. Specifically, YOLO can be trained using different input sizes and can also use a different input size during inference. In both cases, it is somewhat surprising that the accuracy first increases and then decreases when increasing the resolution. The reason for this result is that higher resolution makes large objects harder to detect. Besides, we also found that a model trained and tested at a high resolution performs better than a model that use a high resolution for inference only. While there are many possible choices for image preprocessing, we consider 2 common enhancement techniques, deblurring and contrast enhancement, in this project because there are many blurred and low-light images in the dataset. Due to the absence of a satisfying deblurring tool, we sought to determine the upper limit of improvement in accuracy if deblurring is employed by comparing the performance on artificially blurred images and their original sharp counterparts. Although an increase of up to 2 points in mAP can be obtained, deblurring has limited help for the overall performance because only a small percentage of the images are blurred. Contrary to expectations, this study found that contrast enhancement leads to worse performance even though the image quality is enhanced visually to human. The reason for this is not clear but a possible explanation is that there are a large number of images in low-contrast condition in the training set and the model may have learned a good way to extract features from these images while the enhanced images have not been seen during training.

The second research question in this project sought to compare YOLO with RetinaNet and determine which is a more suitable detection engine for our application. Our unanticipated finding was that RetinaNet did not demonstrate any advantage over YOLO in either accuracy or speed though it is claimed to have much better accuracy.

To demonstrate the effectiveness of our work, we developed a final model that takes advantage of several optimizations learned from previous experiment, such as the configuration of resolution, detection scales, anchors and handling class imbalance, and finally achieves very impressive results on the Road Object Detection challenge.

In conclusion, the insights gained from this study may be of assistance to engineers who are building object detection systems for a real-time application such as self-driving car.

6.2 Limitations

We also acknowledge that there are a number of limitations in our approach. First of all, readers should bear in mind that the study is based on YOLO and this may affect the validity and usefulness of our findings in a different context. Secondly, since our work focus on the accuracy/speed trade-off, it is beyond the scope of this study to improve the algorithm itself, such as redesigning the network architecture or loss function, which may lead to huge improvement. Lastly, the limitation of resources does not allow exhaustive experiments. For instance, experiment 5.4 could train YOLO using a broader range of resolutions and experiment 5.7 should train the interim models longer until they all converge to the optimal point.

6.3 Future Work

This project provides the following insights for future research.

Since there are many other choices of modifying the input images other than resizing and the two image enhancement approaches explored, a natural progression of this work is therefore to assess the effects of a wider range of image preprocessing techniques during inference.

In order to improve the accuracy significantly, a further study with more focus on improving the model is suggested. Despite the promising result of our final model on BDD test set, there is still potential to further improve the accuracy of YOLO. Currently, YOLO struggles with the accuracy of bounding box prediction which hinders its mAP at a strict IoU like 0.75 [8]. Further research should be undertaken to investigate the reason and find a good solution.

Several questions still remain to be answered and further experimental investigations are needed to figure out why RetinaNet fails to outperform YOLO in accuracy and why contrast enhancement leads to worse performance.

Last but not least, we believe a python implementation of YOLO will facilitate future research a lot because the current YOLO is written in C, making it very hard to do customization. There are a number of attempts to rewrite it using common framework like Pytorch but they have been unable to reproduce the same results of YOLO due to bad implementation or hyperparameter tuning.

6.4 Things That Did Not Work

It may be worth mentioning some things that we tried but did not work.

GAN (Generative Adversarial Network) [40] has achieved great success in image super-resolution [41] and can also handle image downscaling [42]. We intended to use GAN as a downsampling method to reduce the image resolution before feeding into the object detection network. A specialized GAN in place of a traditional resize method such as Bicubic might help to improve the performance of object detection. However, images can only be downscaled or upscaled by a factor of 2, 3, 4, etc due to the limitation of CNN while most object detectors requires a specific resolution that is likely to mismatch the size of the scaled image.

One-stage or dense object detectors usually predict whether an object exists at each spatial position in the image while only a few locations have objects. Lin et al. argue that the large class imbalance that occurs in this background and foreground classification hinders the performance of dense object detectors because the loss of a great number of easy examples will overwhelm the loss of rare hard examples. They propose a novel focal loss to address this problem and demonstrate it enables RetinaNet to achieve very promising results. Focal loss is a variant of binary cross entropy that down-weights well classified examples (e.g., prediction is 0.95 while target is 1) and thus reduces the loss from easy class in YOLO. Inspired by RetinaNet, Redmon and Farhadi tried to apply focal loss to the objectness prediction of YOLO but it did not work. They explained this result by the fact that, unlike RetinaNet, class prediction error does not incur any loss if there is no object. Despite this, focal loss might help to mitigate the large class imbalance in our dataset. We tried to incorporate focal loss into YOLO for object classification but the mAP dropped by 1.4 points. It may be that the imbalance in our dataset is not comparable to the foreground-background class imbalance.

Bibliography

[1] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *ACM SIGPLAN Notices*, volume 53, pages 751–766. ACM, 2018.

[2] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[3] 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.

[4] 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.

[5] 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.

[6] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

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

[8] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[9] Khalid Ashraf, Bichen Wu, Forrest N Iandola, Mattthew W Moskewicz, and Kurt Keutzer. Shallow networks for high-accuracy road object-detection. *arXiv preprint arXiv:1606.01561*, 2016.

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[11] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[12] Christoph H Lampert, Matthew B Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.

[13] 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.

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

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[16] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[17] 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.

[18] Bharat Singh and Larry S Davis. An analysis of scale invariance in object detection snip. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3578–3587, 2018.

[19] Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In *Advances in Neural Information Processing Systems*, pages 9310–9320, 2018.

[20] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. *arXiv preprint arXiv:1812.01600*, 2018.

[21] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. *arXiv preprint arXiv:1901.01892*, 2019.

[22] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[23] Mohamed Aladem, Stanley Baek, and Samir A Rawashdeh. Evaluation of image enhancement techniques for vision-based navigation under low illumination. *Journal of Robotics*, 2019, 2019.

[24] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[27] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[28] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[29] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge & Data Engineering*, (1):63–77, 2006.

[30] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[31] Chris Drummond, Robert C Holte, et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11, pages 1–8. Citeseer, 2003.

[32] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[33] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *arXiv preprint arXiv:1804.06215*, 2018.

[34] Raman Maini and Himanshu Aggarwal. A comprehensive review of image enhancement techniques. *arXiv preprint arXiv:1003.4053*, 2010.

[35] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3883–3891, 2017.

[36] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiří Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8183–8192, 2018.

[37] Zhenqiang Ying, Ge Li, Yurui Ren, Ronggang Wang, and Wenmin Wang. A new image contrast enhancement algorithm using exposure fusion framework. In *International Conference on Computer Analysis of Images and Patterns*, pages 36–46. Springer, 2017.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[39] Qijie Zhao, Tao Sheng, Yongtao Wang, Feng Ni, and Ling Cai. Cfenet: An accurate and efficient single-shot object detector for autonomous driving. *arXiv preprint arXiv:1806.09790*, 2018.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[41] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[42] Adrian Bulat, Jing Yang, and Georgios Tzimiropoulos. To learn image super-resolution, use a gan to learn how to do image degradation first. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 185–200, 2018.

Appendix A

Supplementary Materials

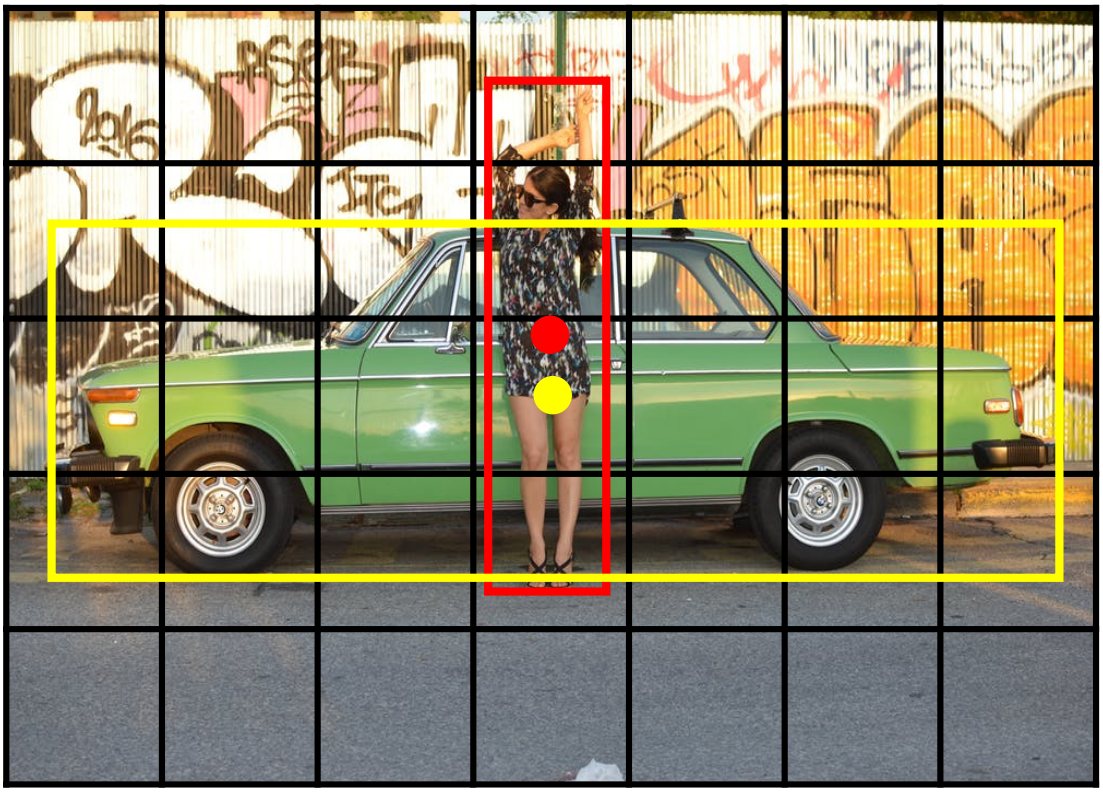


Figure A.1: Two objects are assigned to the same grid cell on the feature map. Two anchors with aspect ratios of 1:3 and 3:1, for example, will fit them better than just one squared anchor. This picture is used under Pexels License.

Algorithm 1: Non-Max Suppression

Input: Bounding box predictions from all yolo layers. Each of them has 4 bounding box coordinates (x_1, y_1, x_2, y_2), classification probabilities *class_probs* for all classes and an objectness score obj indicating whether there is an object.

Output: Final bounding box predictions.

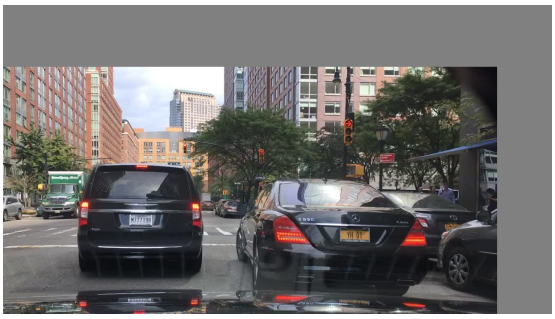
Initialize: $conf\_thresh = 0.005, \quad nms\_thresh = 0.45$

foreach *prediction P* **do**
  foreach *class c* **do**
    $P.score[c] = P.obj * P.class\_probs[c]$;
    if $P.score[c] < conf\_thresh$ **then**
      $P.score[c] = 0$;
    end
  end
end
foreach *class c* **do**
  sort all predictions by $score[c]$;
  foreach *prediction A* **do**
    if $A.score[c] > 0$ **then**
      Output A ;
      foreach *prediction B after A* **do**
        if $iou(A,B) > nms\_thresh$ **then**
          $B.score[k] = 0$;
        end
      end
    end
  end
end

(a) original

(b) horizontal flip

(c) color adjustment in HSV space



(d) translation

Figure A.2: Different kinds of image augmentation.

category	train	motor	rider	bike	bus	truck	person	light	sign	car
weight	1000	50	50	30	10	5	10	3	5	1

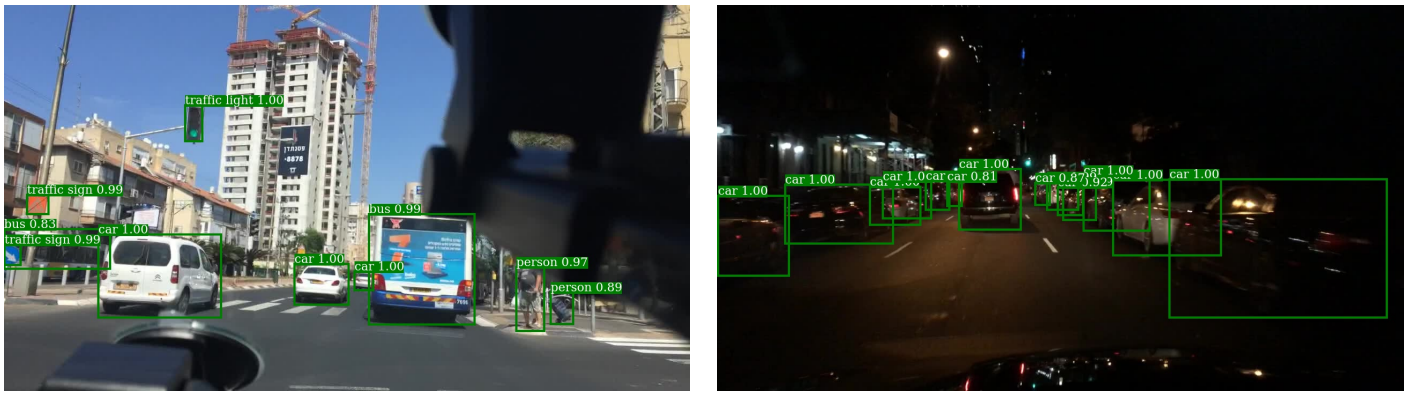
Table A.1: Sampling weights assigned to different classes. The weight of an image is the maximum weight of the classes it contains. That is to say, an image in the training set will appear 1000 times if it contains *train* but just once if it only contains *car*.

level	baseline	final
P_1	——	(8,8)
P_2	——	(16,16)
P_3	(19,19), (44,46), (95,66)	(32,32), (32,64), (64,32)
P_4	(76,160), (164,109), (255,172)	(64,64)
P_5	(233,334), (389,262), (513,442)	(128,128), (256,256), (512,512)

Table A.2: Comparison of anchors in the baseline and final model.

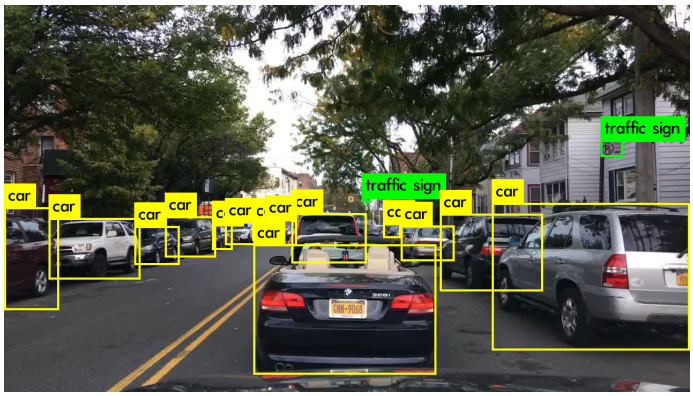
Appendix B

Examples

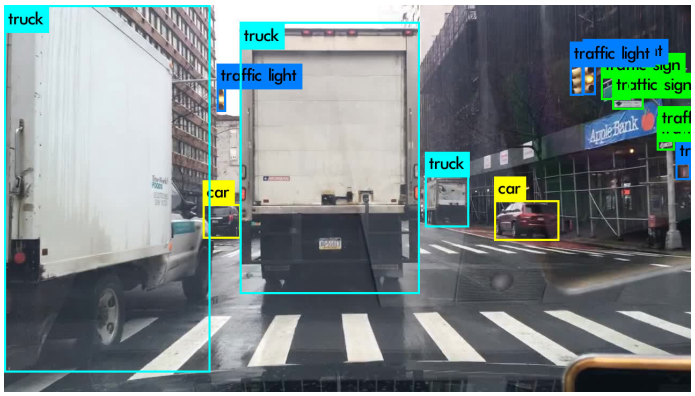


(a) (b)

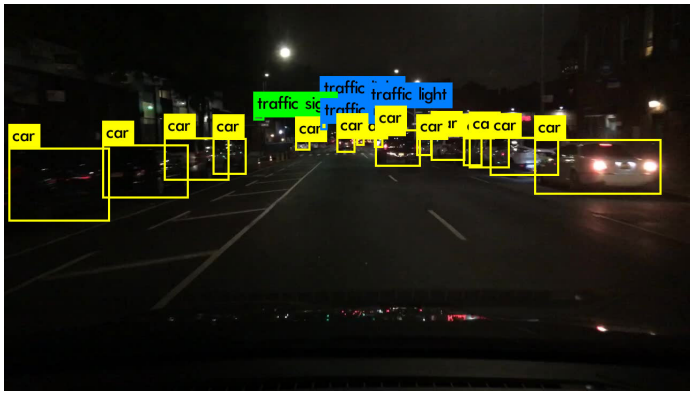
Figure B.1: Output examples of RetinaNet.



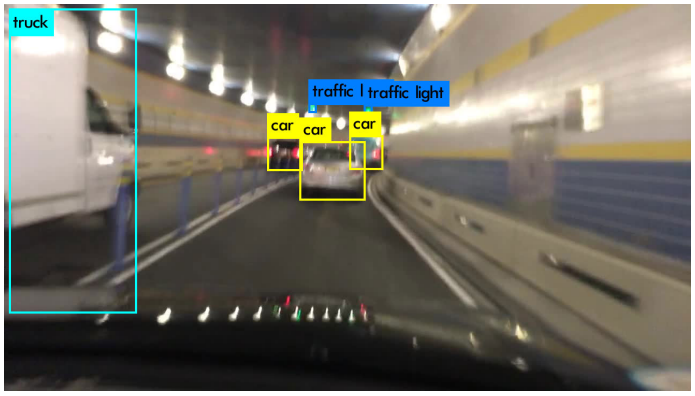
(a) dense objects



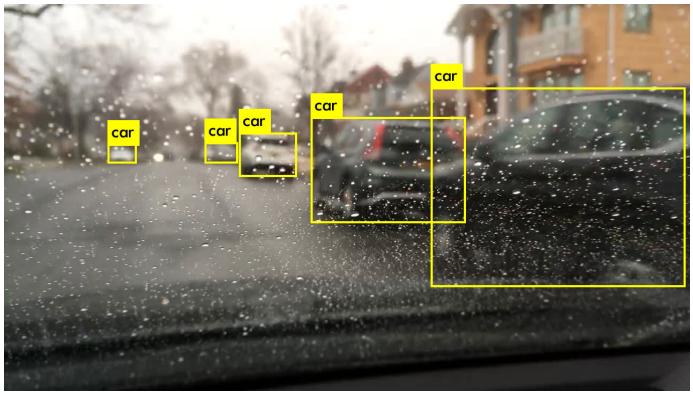
(b) big objects



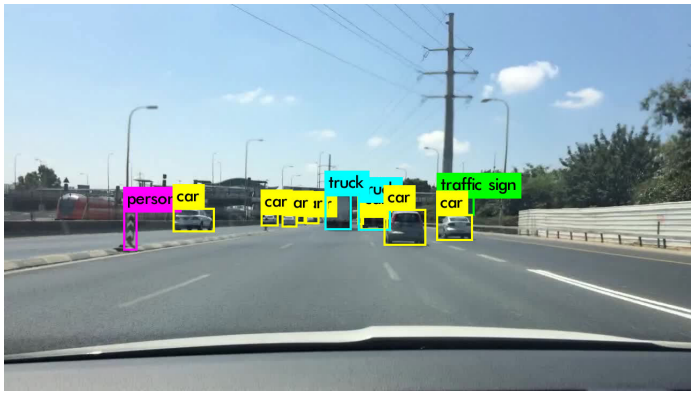
(c) at night



(d) blurred



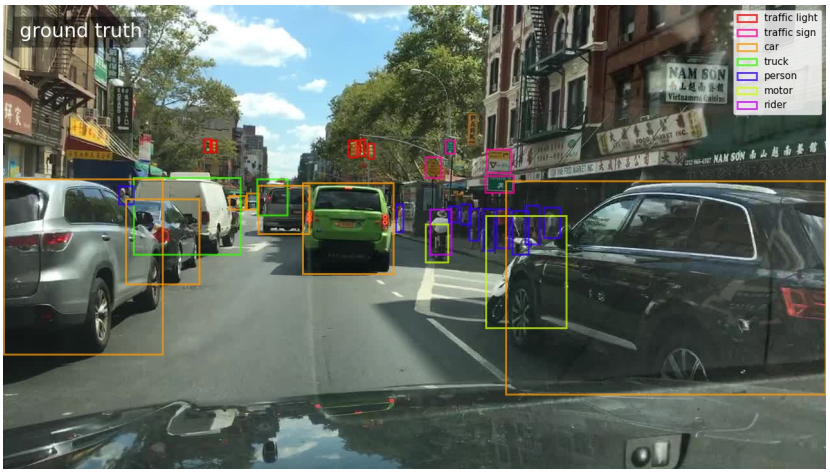
(e) rainy



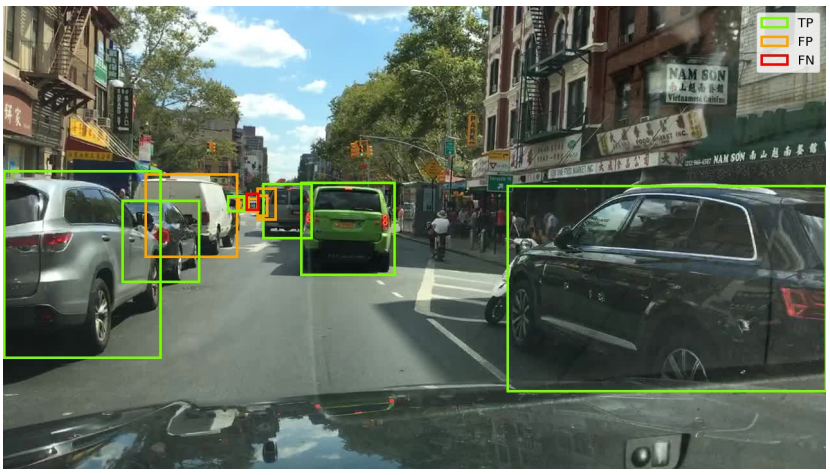
(f) YOLO fails to detect the *train*

Figure B.2: Output examples of the final model.

(a) Detection output



(b) Ground truth



(c) TPs, FPs and FNs in category *car*

Figure B.3: Detection output of the final model versus the ground truth. (c) shows correct, wrong and missed object detections for a specific category *car*.