School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Final Report

Research on Recognition Algorithm Based on YOLO V3

Shuai Wang

Submitted in accordance with the requirements for the degree of <BSc Computer Science>

<2020/2021>

40 credits

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Printed copy of report	*Report*	*SSO (10/05/2021)*
Online report	*PDF*	*VLE(10/05/2021)*
Code	*Github URL*	*Supervisor, assessor (10/05/2021)*

Type of Project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) Shuai Wang

Summary

This project reimplemented YOLOv3 through Torch and get a similar performance on COCO compared with the original author.

Based on the basic implementation of YOLOv3, three improvements have been applied to the model: 1. Improve IoU to CIoU. 2 Add a loss vector to the loss function of YOLOv3. 3. Implement a bayes modifier in YOLOv3 network, and the modifier has been updated three iterations.

In experiment, the effect of CIoU improvement has been proved through light weight training based on PASCAL VOC2007 dataset. In addition, the loss vector also improved a lot on the mAP on light weight training in PASCAL VOC. At last, through test on COCO2014 based on the trained weight by author, the bayes modifier has been proved that it can improve the mAP.

Finally, the main codes have been uploaded to GitHub, and the detailed codes and weight have been uploaded to the cloud server named Featurize.

Acknowledgements

Thanks for my supervisor Hou Jin, who instructed the direction for many times when I feel confused. Furthermore, she also assigned postgraduate Kang Pinging to answer the technical details.

In addition, thanks for Tom Kelly. As my assessor, he provided many useful feedbacks on my mid-term report, and also gave me some valuable advices for the report structure.

Contents

Chapter 1
Introduction

1.1 Project Background

In the last decade, the ubiquitous need of intelligent life such as video surveillance, robotics, and self-driving have triggered a significant boom of research in the field of computer vision. As the core of all these applications, the object detection, defined by determining both location and category for each object, has gained a great momentum of research [1] [3]. Due to the remarkable development of neural networks, especially by the application of CNN (convolutional neural network), the object detection systems have achieved excellent performance.

Since the born of the Region-CNN, the first CNN-based object detection algorithm, both the speed and accuracy of detection grows extremely fast. For those two-stage detection algorithms, from R-CNN, Fast R-CNN, to Faster R-CNN, the speed increased by tens of times, while the accuracy on Pascal VOC and COCO has also increased a lot [3]. As for those one-stage detection algorithms, such as SSD and YOLO (from v1 to v5), the speed grows even more quicker. To be more specific, the fastest yolo, yolo v5 s can detect an image in less than 5ms with an accuracy of more than 35%mAP in COCO dataset.

From yolo v1 to yolo v3, those three awesome algorithms are designed by same author named Joseph Redmon. While v4 and v5 are implemented by two independent engineers based on yolo v3 in 2020. In addition, although those state-of-art models are so strong and fast, the training of the network is still a massive work including outstanding GPU, huge dataset and a quite long time. Thus, in this project, the main aim is study and reimplementation yolo v3, then to improve it in some aspects.

1.2 Aim

The main purpose of this project are to improve YOLOv3 without experiencing heavy training . Specifically , it can be divided into two types :

1 . Improve mAP under the premise of insufficient training .

2 . Improve the mAP on COCO dataset of the model through the original author's network parameters .

1.3 Deliverables

1. Project source code except network weight, since GitHub does not allow huge single file (GitHub)

2. Report (PDF as this document)

Chapter 2
Project Planning and Management

2.1 Project plan

To achieve the project of improving YOLOv3, there are five steps in total: theory study, YOLOv3 reimplementation, prompt theoretical improvement, implement improvement, deployment and summary.

Theory study: First learn the specific requirements of the object detection task, then study the history and change of object detection algorithm in recent years, then learn the main features and changes from the YOLO series, and finally focus on learning YOLOv3. Time plan: 2020/9-2020/12

YOLOv3 reimplementation: Find the reimplementation code of YOLOv3 on GitHub, configure the relevant CUDA, CUDNN, Pytorch, TensorFlow and other environments, run through and debug the training, testing, and detection functions, and convert the PASCAL VOC data set into the required format . Time plan: 2020/11-2020/1

Prompt theoretical improvement: Research the related improvements of YOLOv3, find the way to improve the YOLOv3 framework, and establish the relevant theoretical basis. Plan time: 2021/1-2021/4

Implement improvement: Implement the proposed improvement to the code level, record important experimental results, and save important implementation versions. Planning time: 2021/3-2021/4

Deploy and summary: Summarize the results of the previous experiment, match the best improved model in the experiment, and upload the mature code.

2.2 Version control

GitHub was chosen as the tool for version control. However, in experiment, each version implementation are based on different weights (more than 200MB for one single weight file), which is not allowed for uploading to GitHub. Furthermore, due to the project was implemented in many different devices, the cloud server named featurize was chosen as the place for version control.

2.3 Risk Management

Since YOLOv3 is a quite new, complex and unfamiliar model, The risk management is quite indispensable. Those risks have been considered.

1. Fail for reimplementing YOLOv3: Study the basic idea of TensorFlow, Torch and Keras simultaneously, find different version of reimplementation of YOLOv3, which can clearly decrease the risk of failing.

2. Cloud server crash: For each new huge update, download the project to the local computer

3. Improvement is invalid (happened but avoid): Whenever a theoretical upgrade is fully proposed, first search the relevant literature, and then ask the supervisor about the feasibility

2.4 Actual process

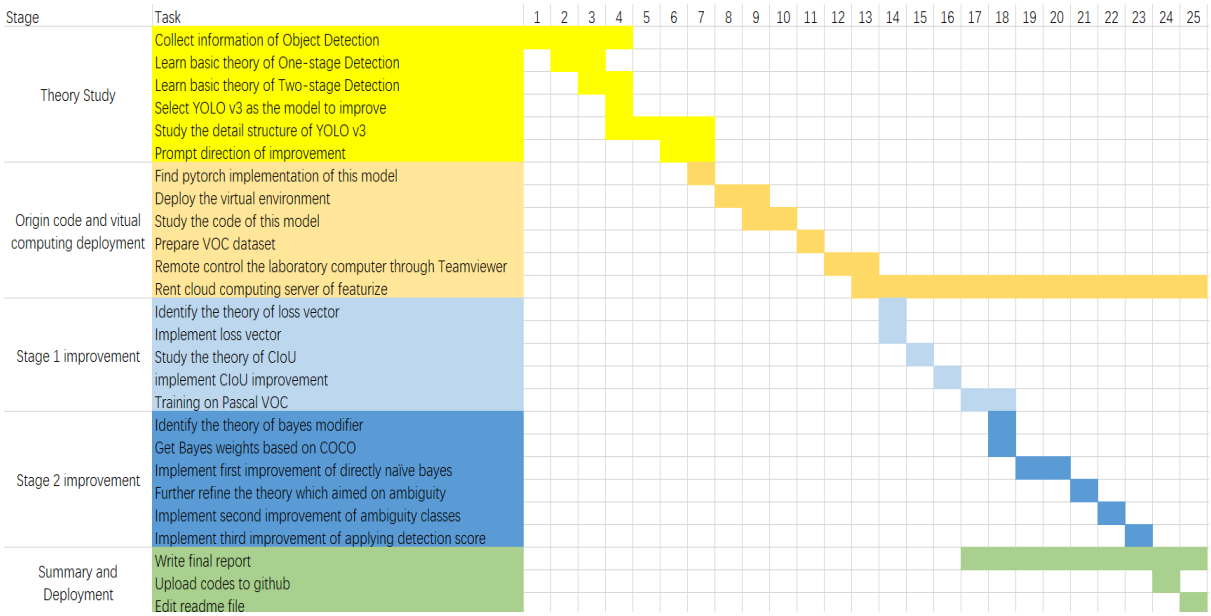

Fig 1.1 Project Time Scale

Chapter 3
Background Research

3.1 Architecture of CNN

Convolutional Neural Network (CNN), as one of the most representative deep learning-based models [4], is a common feedforward neural network for image classification [5][6]. In general, the core of CNN is extracting key information from images via lots of operations such as convolution and pooling. A brief representation of CNN has been shown in figure 3.1

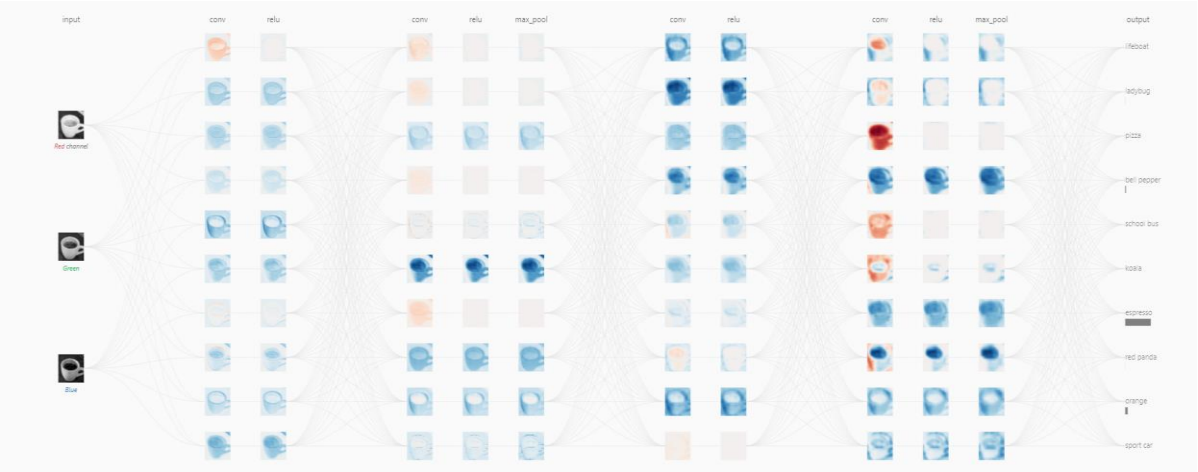

Figure 3.1 working sample of CNN [20]

The first process of CNN is transforming image to numbers. To be more specific, for different color channels (e.g., RGB), those pixels with color can be transformed into the 3-D matrix, such as 416*416 with RGB image can be modified to 3*416*416 matrix while each values of matrix represents the brightness of RGB.

The next part is normally many layers of alternate convolutional and pooling implemented by many neurons. Each neuron is connected with a group of previous layer's adjacent neurons based on the receptive field.

Convolutional layer is mainly aimed to extract the information [7]. In mathematical concept, convolutional filters multiply the weight and output of neurons of prior layer, then add a bias weight to calculate the output of each neuron. Next, the sum of all the neurons of a filter is the value(input) of next layer's neuron. In addition, activation function (usually ReLU) always was used to modify the output of convolution layers, while LeakyReLU (negative part multiplied by 0.1, non-negative part no change) has been used in this project.

Pooling layer is mainly aimed to summarizes the most important value of prior layer, such as max pooling and average pooling. Different types of pooling calculated in different ways, but

usually it is a very efficient way to reduce the amount of computation. Furthermore, zero padding has been applied to this project to put zeros in those un-computable positions.

Residual learning is also a key role in those complex neural networks. Through connecting layers in different depth, a lot of unnecessary computations can be ignored. Thus, the network can be built deeper with less influence of degradation.

Based on CNN, the ability of computer vision to obtain information has been greatly improved, which provides the basis of the CNN-based target detection algorithm starting from R-CNN [3].

3.2 Two-stage object detection

Two-stage object detection methods like traditional object detection pipeline. Firstly, for a given image, it generates region proposals (RP). Secondly, for each RP, classify the object categories. Those region proposal-based methods mainly include R-CNNs (R-CNN [8], Fast R-CNN [2], Faster R-CNN [9], Mask R-CNN [10]), SPP-net [11] and region-based fully feature pyramid networks (FPN) [12]. In addition, the most typical two-stage detection models R-CNN, and the model Faster R-CNN that connects YOLO and two-stage detection, would be introduced.

3.2.1 R-CNN

Based on the application of CNN, Girshick et al. prompt R-CNN [8], which had a significant improvement on object detection. To be more specific, it obtained a mean average precision (mAP) of 53.3% on PASCAL VOC 2012, while the previous best method (DPM histograms of sparse codes [13]) is 30% lower than R-CNN. The mainly structure consisted of three modules has been shown in the figure 3.2

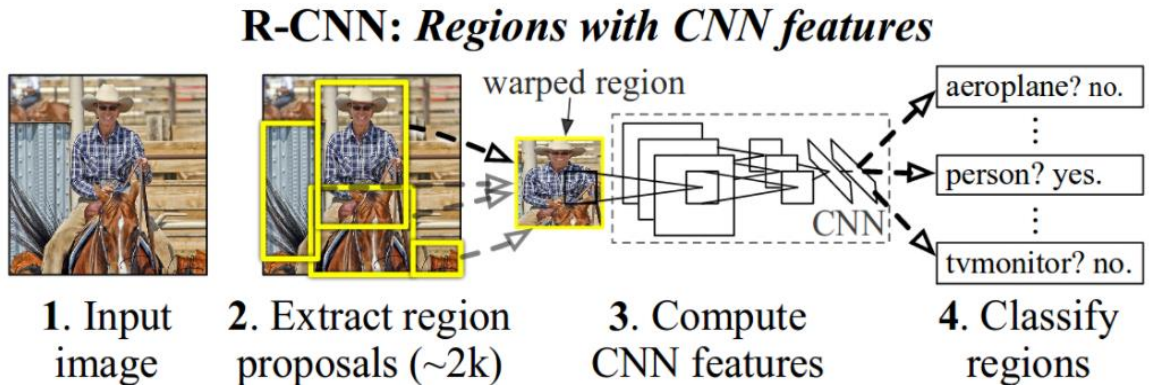

Figure 3.2 architecture of R-CNN [8]

The first module is Extract Region Proposal (RP) Generation: For each image, through selective search [14], 2000 RP could be generated. The second part is calculating CNN features: For each RP, modified the size into 227-227, then extract a 4096 dimensions

vector through Alexnet. Thirdly, based on the SVM classifier, outputs the result including bounding boxes position and class. At last, by using greedy non-maximum suppression (NMS), generate the final result.

Although the R-CNN was a state-of-art object detection algorithm, there still exists some problems:

1. Since Alexnet can only deal images with size of 227*227, all of the RP should be modified to this size, which costs a lot.

2. The selective search is very time-consuming, actually each image takes 2 seconds.

3. The CNN is required to deal every single PR, which costs significant time.

3.2.2 Faster R-CNN

Faster R-CNN is the most famous and powerful two-stage object detection method since the born of R-CNN [9], published by Ren et al. The most important innovation of R-CNN is applying region proposal network (RPN), which makes it possible to achieve the goal of completing all tasks in one convolution. In addition, due to this function, Faster R-CNN can also be seen as a bridge that connects one stage and two stage target detection algorithms. Figure 3.3 represents the architecture of Faster R-CNN

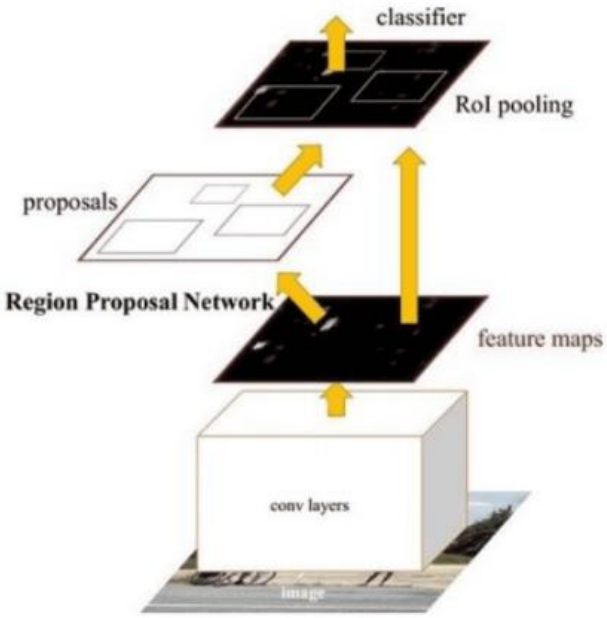

Figure 3.3 architecture of Faster R-CNN [9]

The first part of Faster R-CNN is the convolutional layer. Rather than applying CNN for each RP like R-CNN, the feature maps returned by this CNN would be used in both RPN layer and RoI pooling.

The next part is RPN, which is aimed to generate RPs. To be more specific, in this layer, for those anchors, softmax would be used to judge positive or negative. Then through bounding box regression, it modified anchors to obtain accurate proposals.

The last part is RoI and classification. In this part, prior proposals and feature maps would be dealt to generate proposal feature maps. Finally, for those proposals, the feature maps would be used to calculate the categories, and regression to the final position.

As a result, Faster R-CNN can get 78.8% mAP on PASCAL VOC 2007 Test Set. In addition, it can get 42.7% mAP on MS COCO [9].

3.3 One-stage object detection

Two-stage object detection methods, also known as proposal-based frameworks, due to need generate RP first then classification, are limited in time cost. Thereby, a new kind of framework, one-stage object detection, had been prompt since YOLO and SSD.

In this new framework, those models regard object detection problem as a regression/classification problem. As a result, the time spent by one-stage detection methods is clearly shorter than two-stage methods. Figure 3.4 illustrates the advantages of one-stage detector: they do not need to considering sparse prediction.

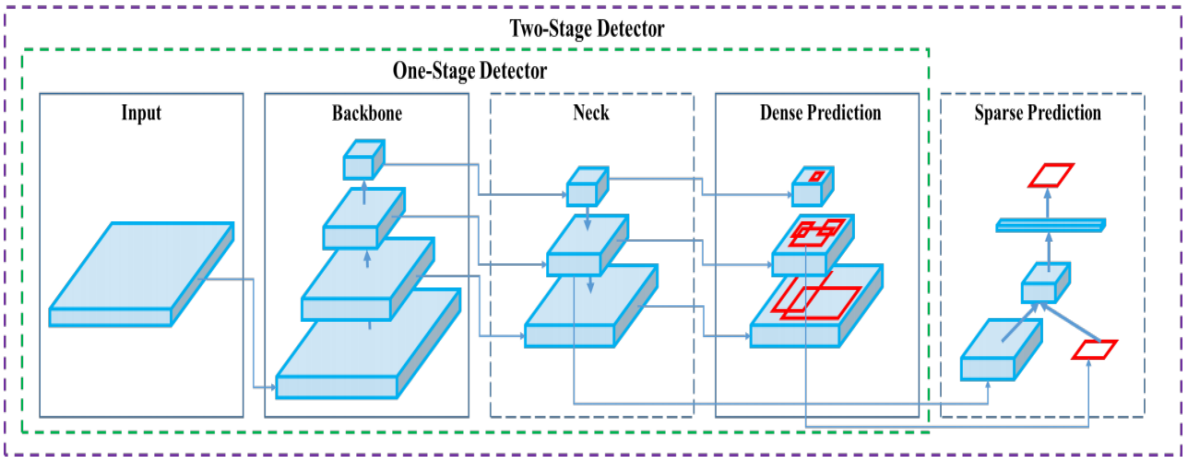

Figure 3.4 Difference between One-Stage and Two-Stage detector [15]

Through all of those one-stage detectors, YOLOs (from YOLOv1 to YOLOv5) are the most representative models (the speed and accuracy of SSD are lower than those of YOLO after the second generation), so the following will introduce the YOLO series

3.3.1 YOLO v1 architecture

YOLO was first published in CVPR on May 9, 2016, with a mAP of 63.4% and a speed of 45 frames per second, became the fastest algorithm at the time. Figure 3.5 compares its performance of accuracy with Fast R-CNN [16].

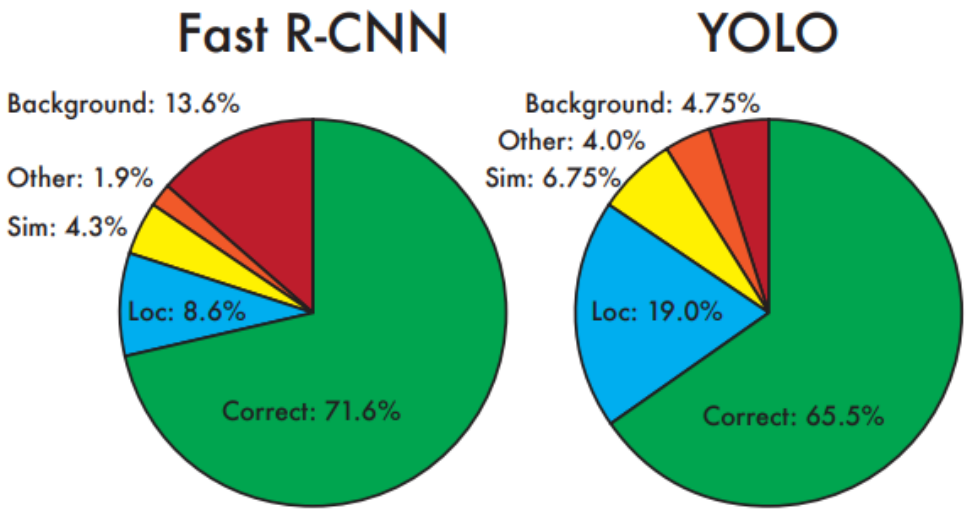

Figure 3.5 Compare accuracy of Fast R-CNN with YOLO in PASCAL VOC [16]

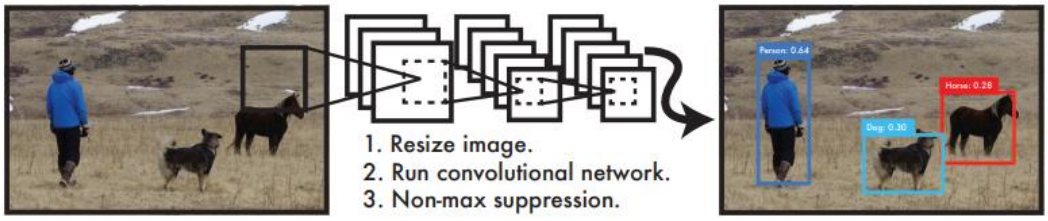

Figure 3.6 YOLO V1 architecture [16]

As shown in the figure 3.6, YOLO first changes the size of the picture to 448 * 448 to realize the unification of the input. Then, the picture is divided into S*S grids, and each grid predicts the position (x, y, w, h), confidence (IoU), and category probability of BB boxes. Therefore, the output dimension is S*S*(B*5+C), and C is the number of categories. No matter how many boxes are contained in the grid, each grid only predicts a set of class probabilities. In the test, the conditional probability and the confidence of the prediction boxes are multiplied to indicate the confidence that each box contains a certain type of object. This score can simultaneously express the category probability and prediction accuracy in the box.

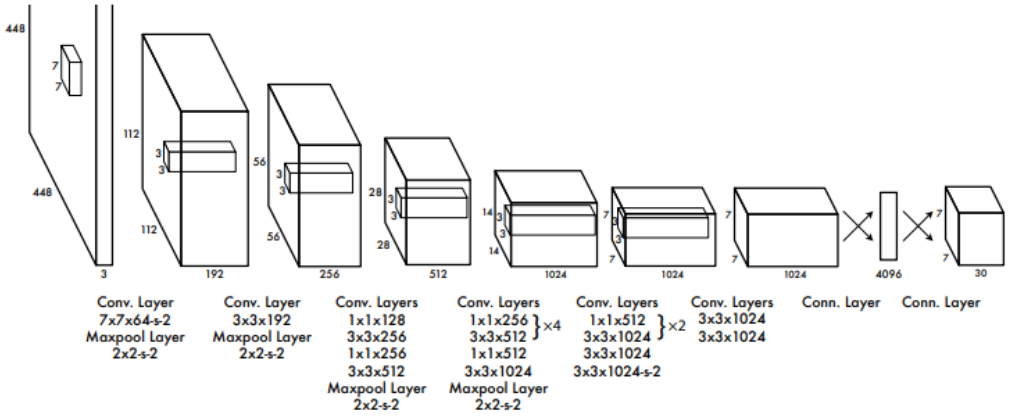

Figure 3.7 YOLO CNN network [16]

As figure 3.7, in convolution training, the convolutional network refers to GoogLeNet 24 convolutional layers, then 2 fully connected layers, and finally output 7 × 7 × 30 to obtain the ideal convolution result.

$$
\begin{aligned}
Loss = {} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
\tag{3-1}
$$

Yolo's loss calculation formula is as formular 3-1. $\mathbb{1}_i^{\text{obj}}$ object appears in cell i, $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the j h bounding box predictor in cell i is “responsible” for that prediction [16]. In addition, x and y represent the position of obj, and w and h represent the position of obj size. It is worth noting that the loss function will only punish those objects that exist.

At the end of the detection, non-maximum suppression is performed. At this stage, under the premise of the specified IoU (0.5 in usual), the candidate boxes of the same category that are higher than the specified intersection ratio will form a set of each other, and only box with max confidence in the set would be shown for the final output.

As the pioneer of single-segment target detection, YOLO has three significant advantages:

1: it is very fast. It can reach 45 frames per second on Titan X, and its simplified version can reach 150 frames per second. Compared with other real-time detection algorithms, its accuracy is twice as high.

2: its detection will consider a wider range of influences. To be more precise, R-CNN divides the picture into multiple pieces for independent convolution, which leads to the lack of surrounding information, while YOLO only convolves the entire picture once, so it can capture more comprehensive information.

3: YOLO has a strong universality. When other artworks are used as input, YOLO can give far more effects than other ordinary models.

But at the same time, YOLO still has some shortcomings:

1: Since there are only two potential boxes for each position, the detection ability for dense small objects is poor.

2: There is still a gap between the overall mAP and the excellent works of the R-CNN series.

3.3.2 YOLO v2 improvement based on YOLO v1

YOLOv2, also known as YOLO 9000, is a more powerful target detection algorithm based on YOLOv1. Compared with yolo v1, its improvement is mainly reflected of three aspects, better, faster, and stronger [17]. The categories of improvements were shown in figure 3.8

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?							✓	✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 3.8 [17]

YOLOv2 can recognize 9000 classes and can reach 76.8 mAP (67 FPS) and 78.6 mAP (40 FPS) on Pascal VOC 2007. Compare with Faster-RCNN (70.4mAP) [9], SSD (74.9mAP) [19], it is clear that YOLOv2 is better than those methods.

3.3.3 YOLO v3 improvement based on YOLO v2

As the cornerstone of this project, YOLOv3 was released in 2018, with the powerful recognition accuracy and speed shown in figure 3.9 [19], it has become one of the best target detection algorithms.

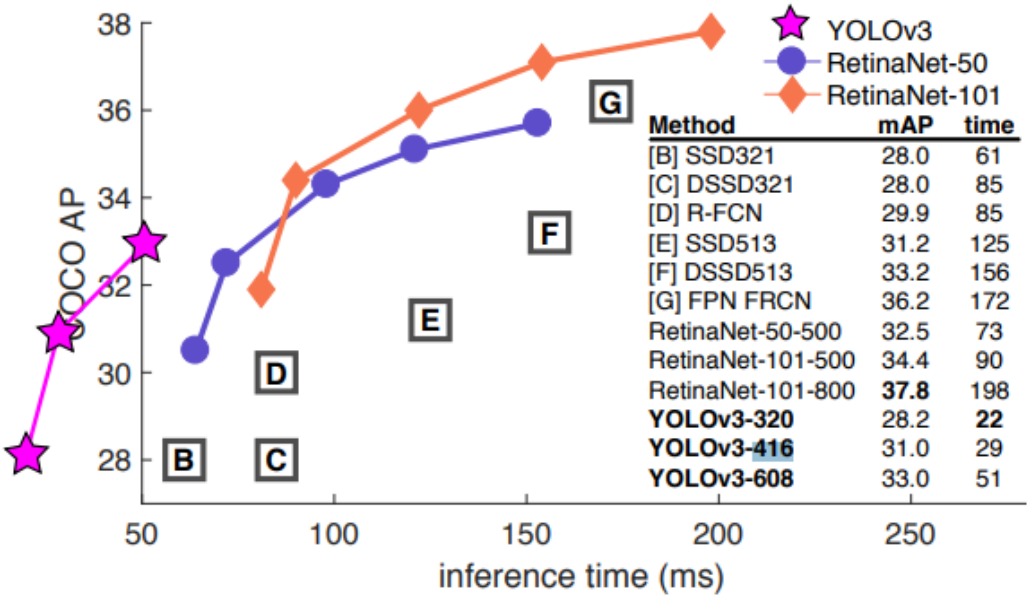

Figure 3.9 [19]

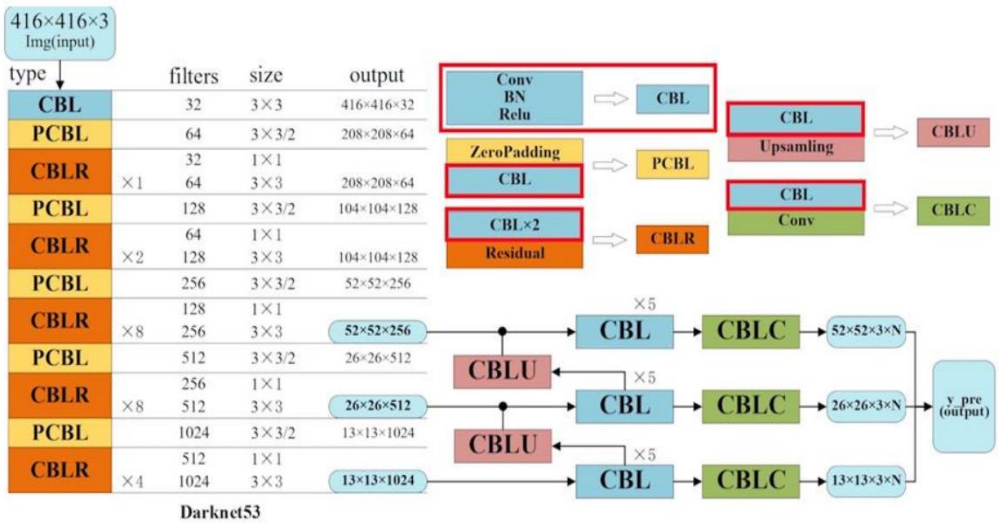

Figure 3.10

YOLOv3 uses Darknet 53 as shown in figure 3.10 as its backbone network. After a total of 53 layers of convolutional networks, three types of results (52*52+26*26+13*13) are output, and larger-scale detection will rely on small-size upsampling. And for three sizes of boxes, three rectangular (bottom<height, bottom=height, bottom>height) boxes are used for object detection at each position. In addition, in the real world, an object may belong to multiple categories (like man and person), YOLOv3 adopts logistics regression instead of SoftMax regression.

3.4 Other improvement for YOLO v3

Because the complexity of the network structure of YOLOv3 is greatly improved, YOLOv3 tiny had been proposed. As figure 3.10 showing, the model uses darknet 19 as the backbone network and cancels the output of the maximum receptive field. As a result, it can reach 33 maps on the COCO data set.

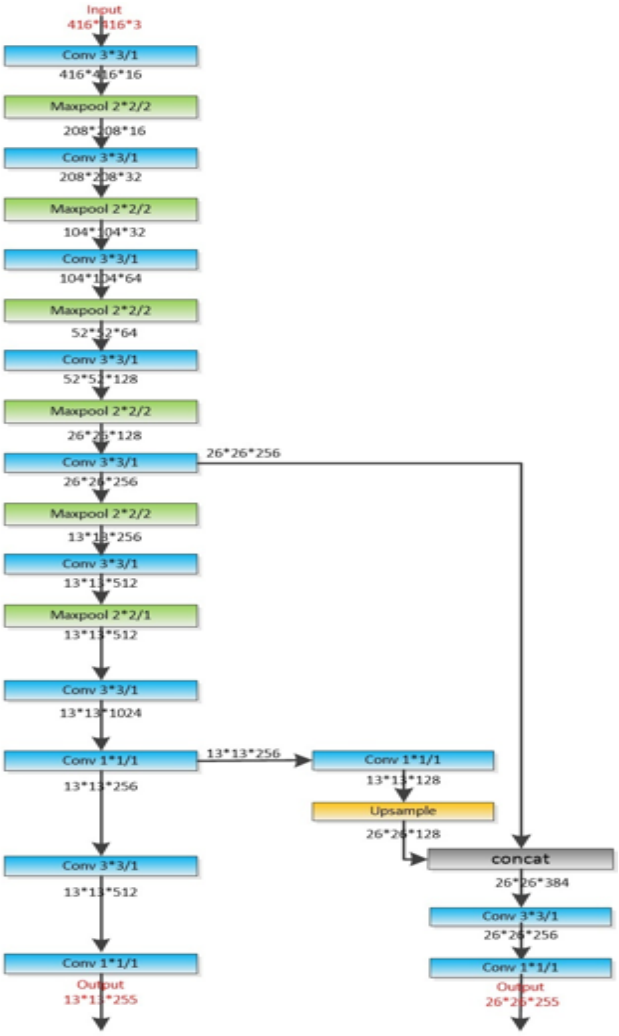

Figure 3.10

Chapter 4
Theorical Implementation

4.1 Analysis of potential improvement

On the whole, YOLOv3 is a powerful and fast target detection. However, due to the large size of its backbone network itself, and the COCO data set originally used by the author is extremely large, about 20GB, it is trained on most GPUs. The time is extremely long (for GTX 1060, an epoch takes about six hours), so it is a very valuable problem to make it more targeted to complete better training under a lower configuration.

4.1.1 IoU

In the workflow of the target detection algorithm, defining the gap between the predicted box and the true box is a very important issue. And YOLO v3 chooses IoU [19] to measure the overlap ratio between the generated prediction box and the target box, as formula (4-1) [21].

$$IoU = \frac{|B \cap B^{target}|}{|B \cup B^{target}|} \tag{4-1}$$

In most cases, IoU can reflect the similarity between the predicted box and target box. But in some cases, IoU may work not well, as below:

1. As figure4.1 illustrating, when those two boxes are not adjacent, since there is no intersection part, the IoU should always be zero (obviously closer boxes performance better)

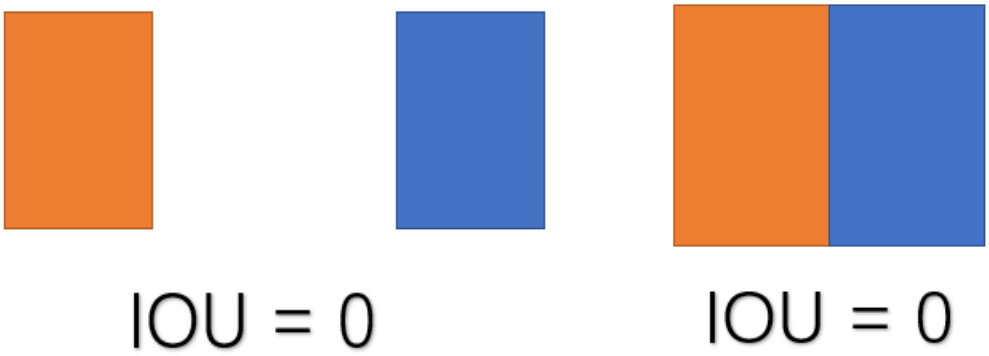

Figure 4.1 In those cases, IoU are both equal to 0 while the latter one is clearly better than former one

2 When those two boxes are different in shape, their IoU can still be same, like figure 4.2

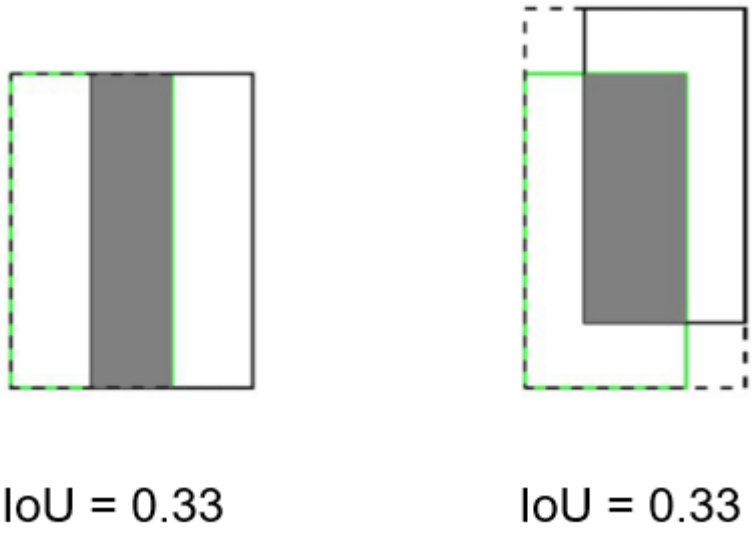

Figure 4.2 two cases for same IoU but different in shape.

Thereby, this project is aimed to find a better metrics instead of IoU.

4.1.2 Class ambiguity

After passing the preliminary training, YOLOv3 can efficiently provide anchor detection, but its classification of different categories is lackluster. At the same time, after increasing the learning rate, the model is prone to encounter the dilemma of overfitting. Therefore, the improvement can begin with the weight of the loss function, increase the weight of the category, in order to guide the category regression through backpropagation.

4.1.3 Object relation

In daily life, there are universal connections and dependencies between objects, but YOLOv3 itself does not capture these dependencies. Therefore, for those objects with ambiguity, if they can be corrected by recalculating the possibility based on other objects, it will undoubtedly improve the accuracy of the classification better.

4.2 Method

4.2.1 IoU improve

This project finally uses Complete IoU (CIoU) [22] to replace IoU as an indicator of the discrimination box. Before CIoU, IoU has undergone three upgrades, namely IoU->Generalized IoU (GIoU), GIoU->Distance IoU (DIoU), DIoU->Complete IoU (CIoU)[23].

1. **IoU->GIoU:** GIoU was first proposed by a Stanford scholar and published in CVPR2019[22], (4-2) is its mathematical formula. Among them, AB is the prediction box and the target box, and C is the smallest rectangle that can contain both A and B.

$$IoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|} \tag{4-2}$$

With GIoU, two boxes that are very far apart and their own boxes are not big, their IoU are 0, while GIoU tends to -1. To be more precise, GIoU is the lower bound of IoU, which can give a reasonable judgment for the boxes that are farther apart, and also give a specified direction for the derivation of gradient descent.

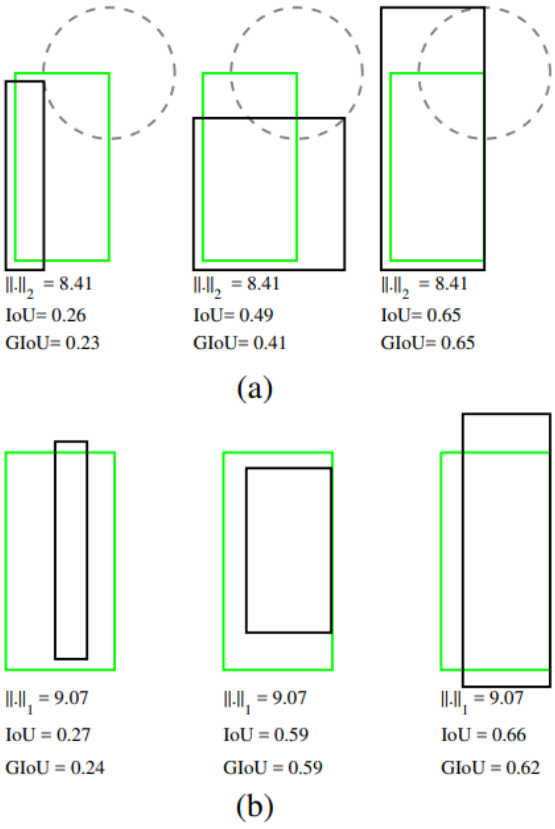

Figure 4.3 example of comparison between IoU with GIoU [22]

As figure 4.3 showing, in both ℓ1-norm distance and ℓ2- norm distance, GIoU are better than IoU [22].

2. **GIOU->DIOU:** DIoU is proposed together with CIoU [23], (4-3) is its mathematical formula. Where b, b^{gt} respectively represent the center points of the two boxes, ρ represents the Euclidean distance of the two points, and c represents the diagonal distance of the smallest outer rectangle that wraps the two boxes

$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2} \tag{4-3}$$

As shown in figure 4.4, when two boxes are contained, GIOU degenerates to IOU, while DIOU can distinguish the similarity between the two according to the degree of inclusion. It can be said that DIoU, which considers the distance between the overlap area and the center point, can better complete the prediction regression of the anchors.

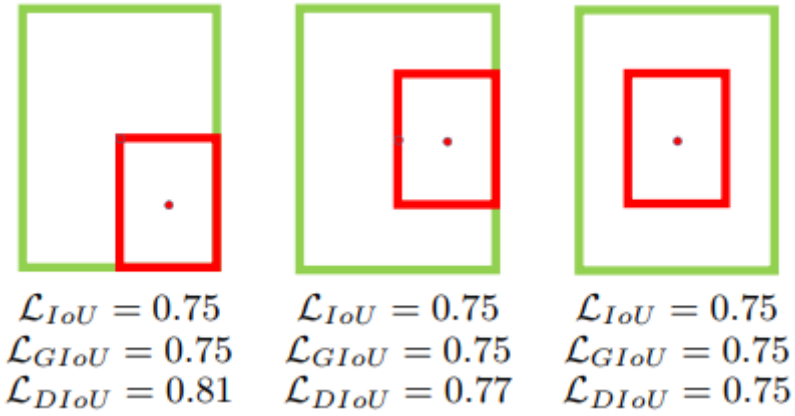

Figure 4.4 example of comparison between IoU, GIoU and DIoU [23]

3. **DIoU->CIoU:** based on DIoU, CIoU further considers the similarity of the shape between boxes, as the formula (4-4) shown, while αv represents the similarity of the shape between those two boxes through arctan.

$$\begin{aligned} CIoU &= IoU - \frac{\rho^2(b,b^{gt})}{r^2} - \alpha v \\ \alpha &= \frac{v}{(1-IoU)+v} \\ v &= \frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\right)^2 \end{aligned} \tag{4-4}$$

4.2.2 Loss vector

The origin YOLOv3 loss function can be divided into three parts: 1 loss of the anchor position; 2 loss of the confidence; 3 loss of the class [19], as (4-5) shown

$$loss = (loss_x + loss_y + loss_w + loss_h) + loss_{conf} + loss_{class} \tag{4-5}$$

In order to make the gradient descent method more flexible to choose the direction of descent without overfitting due to the excessive learning rate, a weight coefficient vector can be added to the original loss as (4-6).

$$\begin{aligned} loss_{new} = weight_{position} * (loss_x + loss_y + loss_w + loss_h) + weight_{conf} * loss_{conf} + \\ weight_{class} * loss_{class} \end{aligned} \tag{4-6}$$

4.3.3 Naïve Bayes modifier

Work position: For each picture, YOLOv3 will give 10647 ((52*52+26*26+13*13)*3) anchors corresponding to the three perception ranges, and there are 85 values inside each anchor (4 positions +1 confidence level + 80 category confidence levels), then classify them, and finally perform non-maximum suppression. The naive Bayesian corrector proposed in this article is to process the 10647*85 data as figure 4.5 shown. The overall process is divided into four steps, 1. Get the conditional probability of each category from the data set. 2. For a picture, decide whether there is any ambiguity. 3. If there is ambiguity, extract sure objects from the picture. 4. Using sure objects as condition, perform naïve bayes calculation on ambiguous classes.

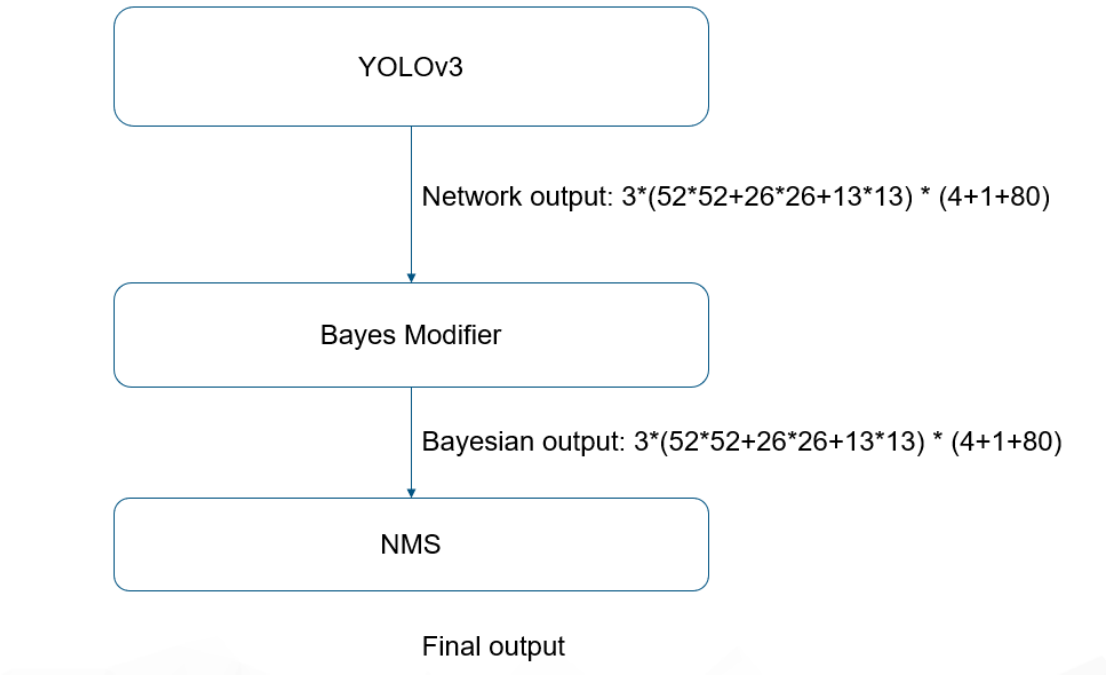

Figure 4.5 Work position of bayes modifier

1. **Get the conditional probability**: Take the COCO data set as an example, extract the information of all the pictures in the training set, then store the conditional probabilities of those 80 classes. At the same time, for each category c of the 80 categories, filter out the pictures that exist in category c, and then calculate the condition of the existence of category c and the conditional probability of 80 categories, and then store the result.

2. **Find ambiguity:** For the 10647 anchors of the picture, first select valuable ones whose confidence is greater than the threshold (0.5), generate set S. Then, for each element s in S, select the most possible class, and other candidate classes that the difference between the most possible class is less than most possible class probability * ambiguous threshold (can be set). If there exists candidate class, that mean there exists ambiguity.

3. **Obtain those surer objects**: For the picture with ambiguity, by calculating the product of the confidence scores of the anchor and the confidence scores of the highest category, all anchor categories with the value higher than 0.9 are retained and placed in set C.

4. **Recalculate the probability of class:** Take the category classification given by YOLOv3 as the prior probability of each category in the B set, and at the same time use the existing class in the C set as the condition, and calculate its value according to the following naive Bayes formula (4-7), as Bayesian corrected score. Finally, the corrected score of the category with the largest score in the B set after correction is compared with the corrected score of the initial largest category, and the largest class is retained to calculate its probability as the final output.

$$P(y_i \mid x_1, x_2, \cdots, x_d) = \frac{P(y_i)\prod_{j=1}^{d} P(x_j|y_i)}{\prod_{j=1}^{d} P(x_j)} \quad (4\text{-}7)$$

4.3.4 Naïve Bayes modifier in three version

During the project, three versions of this modifier has been applied in the proceeded. Those three versions would be mentioned below.

1.In the first version, for all the anchors, recalculate the class probability. To be more specific, for 10647 (3*(52*52+26*26+13*13)) anchors, there is no filter for them. Thus, for each anchor, based on the sure object (threshold = 0.9) as the condition all of the objects would be recalculated.

2.In the second version, the process of finding ambiguity has been implemented. For all of the anchors with ambiguity (threshold = 0.4) as (4-8). In the formula, x_{class} represents the probability score of all the categories except the max one, while S_{sub} is the set that contains all the ambiguous anchors. Then for those ambiguous anchors in S_{sub}, calculate the probability for those candidate classes as the formula of (4-7).

$$S_{sub} = \{\exists x_{class}(\frac{P(x_{class})}{P(x_{max})} > (1 - threshold))|x \in S_{all}\} \quad (4\text{-}8)$$

3.In the third version, based on v2 implementation, the prior possibility has been replaced by the original possibility in in the YOLOv3 output. Through this implementation, more information extracted from YOLOv3 could be considered to have better performance.

Chapter 5
Experimental Implementation

5.1 Experiment environment

5.1.1 Virtual environment for code

The implementation is mainly based on pytorch (a quite powerful deep leaning tool based on python)

The necessary environment including “torch, NumPy, torchvision, matplotlib, tensorboard, terminaltables, pillow, tqdm, imgaug”

Although CPU can work well for detection, CUDA,CUDNN is strongly recommended for GPU training.

5.1.2 Environment for experiment

Due to the training of yolo costs too much, all of the training were running on remote environment provided by Featurize, the settings have been shown in table 5.1.

Table 5.1 environment for training

Type	CPU	GPU
1	Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz	GeForce RTX 3090
2	Intel(R) Core(TM) i3-9100F CPU @ 3.60GHz	GeForce RTX 2080 Ti

The demand of test and detection is quite light. All of the machine in table 5.2 can have good performance.

Table 5.2 environment for test and detection

Type	CPU	GPU
1	Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz	GeForce RTX 3090
2	Intel(R) Core(TM) i3-9100F CPU @ 3.60GHz	GeForce RTX 2080 Ti
3	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz	GeForce GTX 1060

5.2 Experiment based on training

5.2.1 Data preprocess

PASCAL VOC: The dataset chosen is Pascal VOC (Visual Object Classes) 2007, which provides standardized image data sets for object class recognition. And there are 20 classes including: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor. In addition, the dataset consists of 9,963 images containing 24,640 annotated objects.

However, to adapt this dataset into YOLOv3, two kinds of changes are required: data config and data transformation. To config the data, the classes with name were loaded, all of the image and label info paths are recorded for the model. Since the PASCAL VOC data is in XML format, and save the object by absolute position, while this project requires label file in txt format and relative position. Thereby, a python script had been written to recalculate the label files, and the result were shown as figure 5.1.

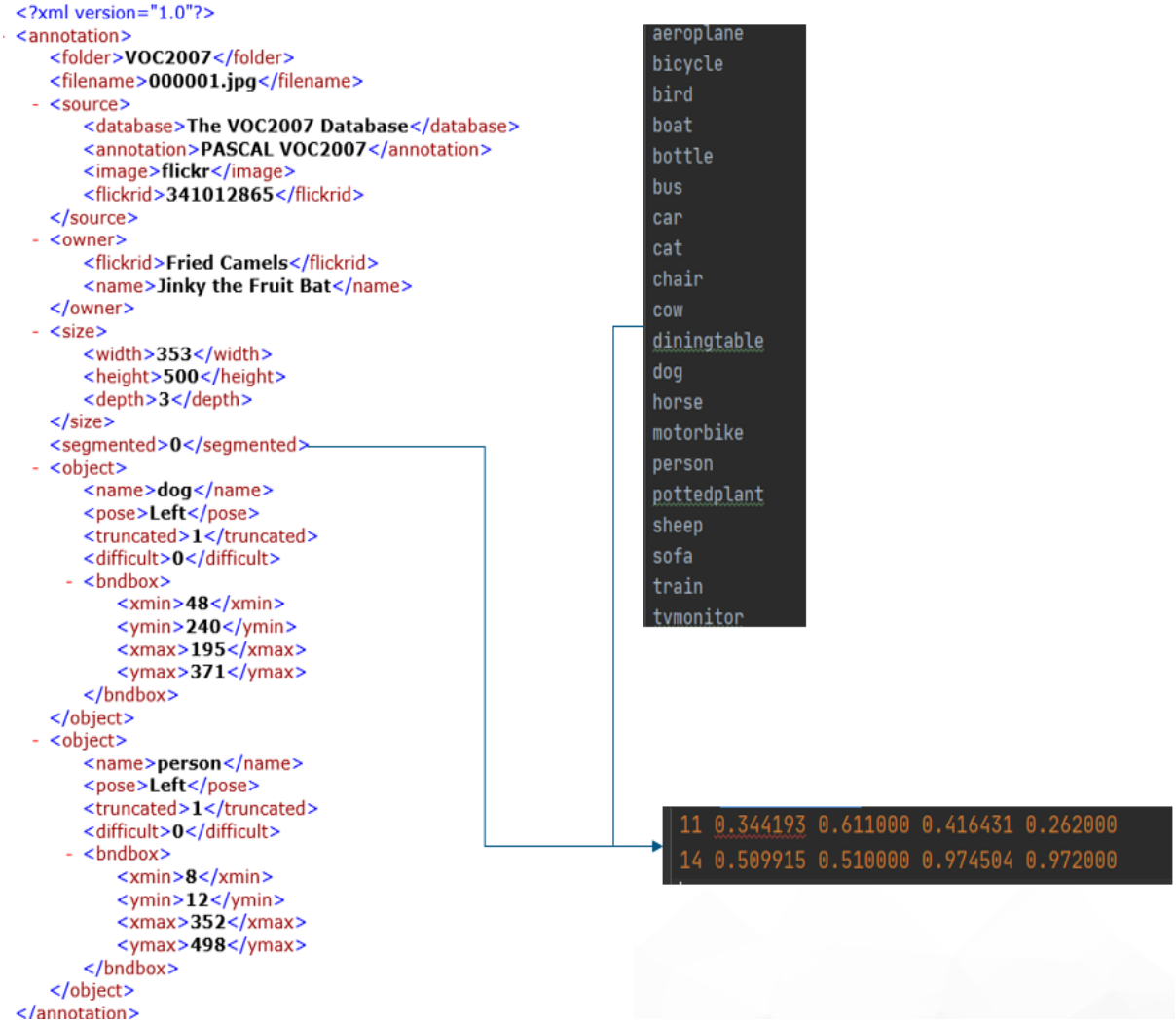

Figure 5.1 transform the PASCAL VOC label

5.2.2 Argument setting

In order to better verify the effect of improvement, the project adopts the method of controlling variables, and the training parameters are fixed to the parameters shown in Table5.3.

Table 5.3 environment for test and detection

Argument	Value
Batch	24
Subdivisions	1
Width	416
Height	416
Channels	3
Momentum	0.9
Decay	0.0005
Learning rate	0.001
Burn in	1000
Steps	400000, 450000
Scales	0.1,0.1

5.2.3 Origin version

The origin version of reimplementation of YOLOv3 is based on Apple Machine Learning engineer eriklindernoren [24]

In training process, for each epoch, the weights would be used to calculate the mAP through test set of PASCAL VOC2007.

However, due to the mAP per epoch curve has severe fluctuations (shown in figure 5.2), a new curve consisted of the average mAP per ten epochs have been recorded (shown in figure 5.3) to illustrate the training process

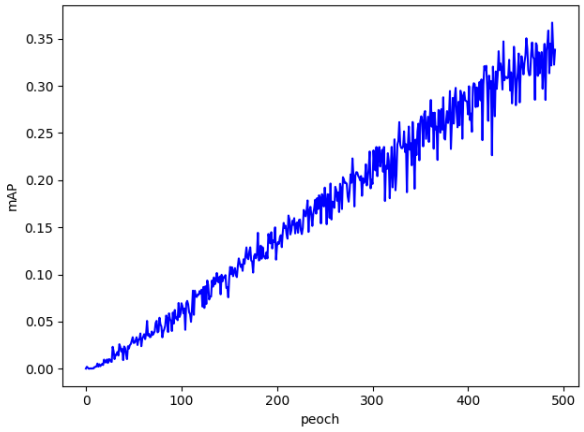

Figure 5.2 mAP per epoch for origin

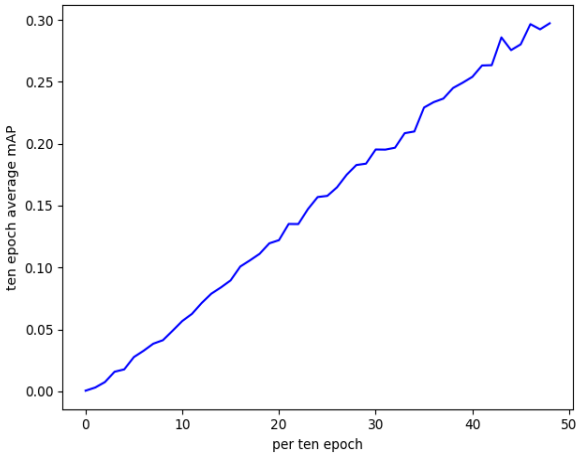

Figure 5.3 mAP per ten epochs for origin

In general, mAP improved to above 0.30, while the best mAP is 0.367 in 496th epoch.

5.2.4 CIoU improve

By applying CIoU to the project instead of IoU, the result was shown in figure 5.4 and figure 5.5

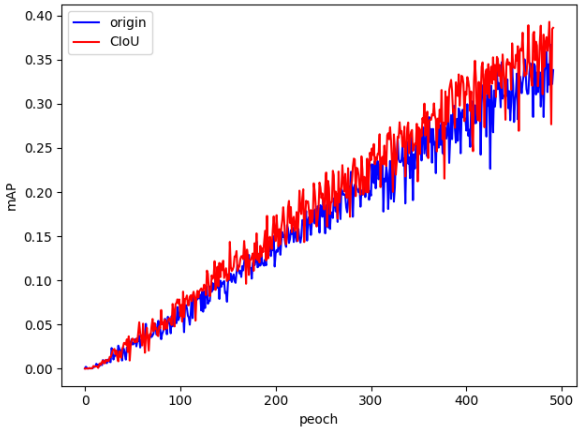

Figure 5.4 mAP per epoch for origin and CIoU improve

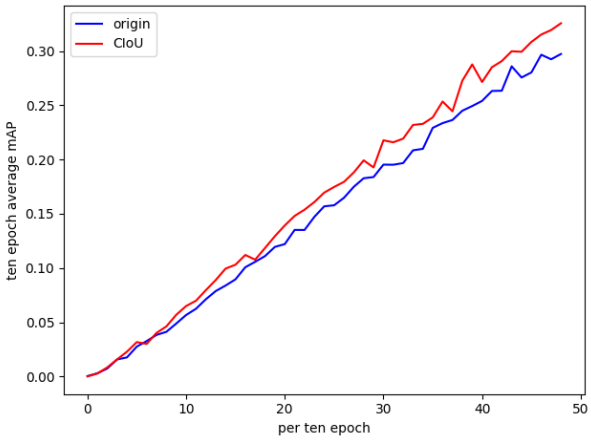

Figure 5.5 mAP per ten epochs for origin and CIoU improve

In summarize, the mAP improved subtle but clear. The best mAP of CIoU version is 0.392 in 495th epoch.

5.2.5 Loss vector improve

By applying loss vector valued of (1,1,3) to the project, the result was shown in figure 5.6 and figure 5.7

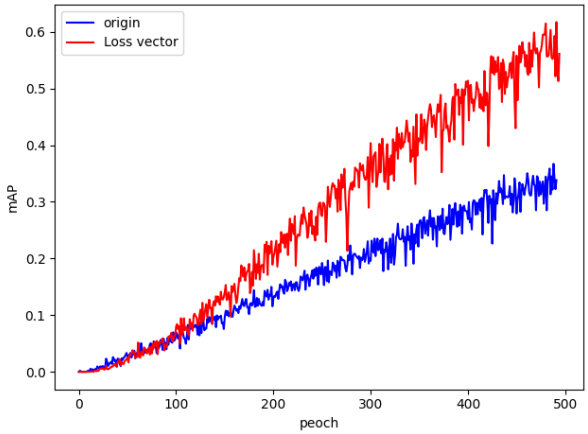

Figure 5.6 mAP per epoch for origin and loss vector improve

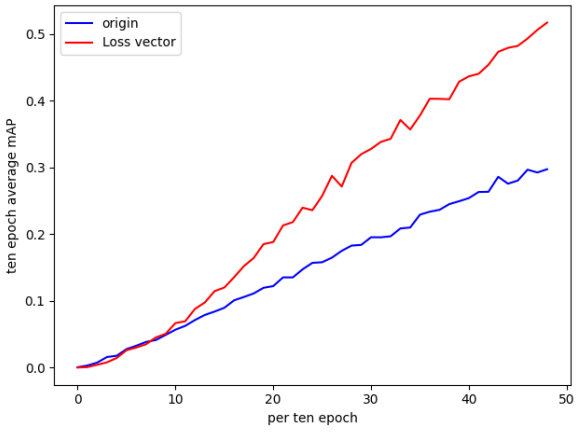

Figure 5.7 mAP per ten epochs for origin and loss vector improve

Overview, the mAP improved quite clear since around 100 epochs (due to first hundred epochs anchor loss is still quite important). The best mAP improved from 0.367 to 0.617.

5.2.6 Combine CIoU and Loss vector improve

By applying both loss vector (1,1,3) and CIoU to the project, the result shown in figure 5.8 and figure 5.9

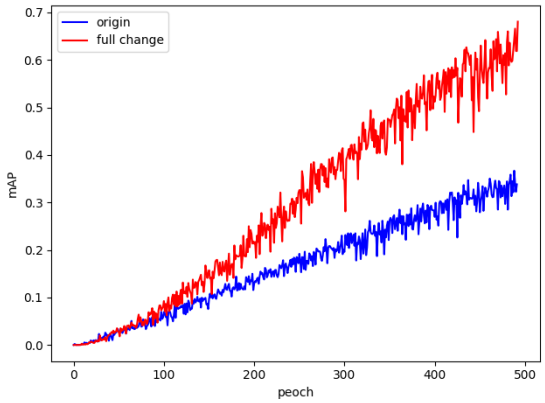

Figure 5.8 mAP per epoch for origin and full-change improve

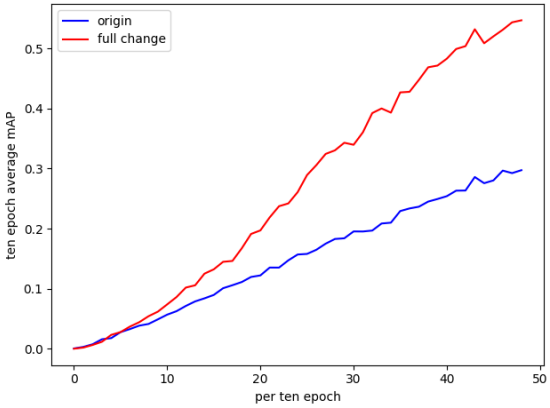

Figure 5.9 mAP per ten epochs for origin and full-change improve

In a word, it is quite clear that the full change version did better than origin version. To be more specific, the best mAP achieved by full change version is 0.680 in 499th epoch, which is 0.313 higher than the best map for origin version

5.3 Experiment based on pretrained weight

5.3.1 Origin version

When there is no change for the model, with the weight provided from the author of YOLOv3, in COCO2014, the mAP is 0.5412, while the ap for each shown in figure 5.10.

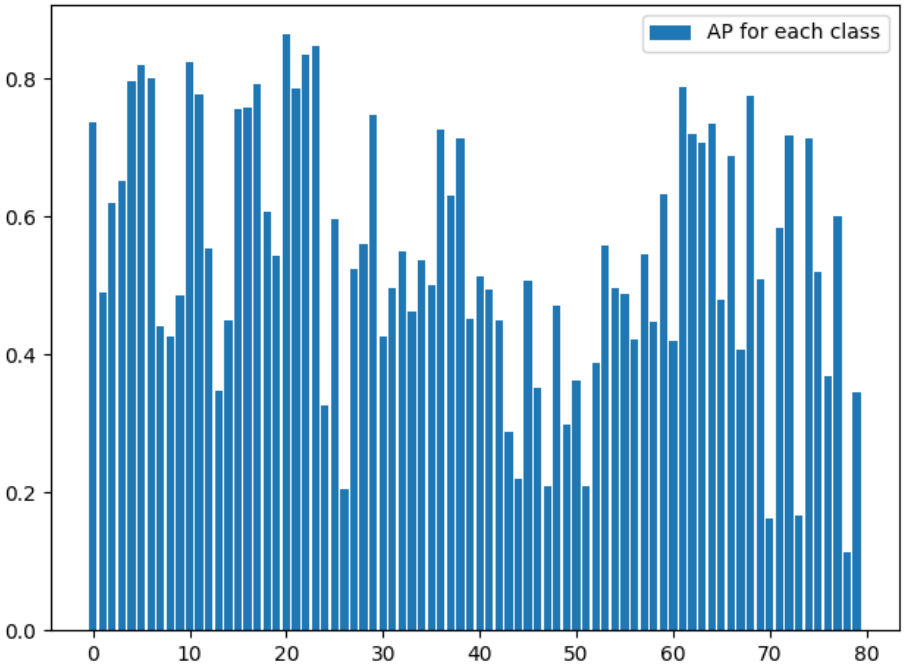

Figure 5.10 AP for each class in origin

5.3.2 Bayes modifier v1

The first time attempt is for all the anchors, by using naïve bayes modifier, recalculate the probability for classes. The mAP increased subtle, from 0.5412 increased on 0.5416. By comparing each AP, the difference between new model with origin model was shown in figure 5.11

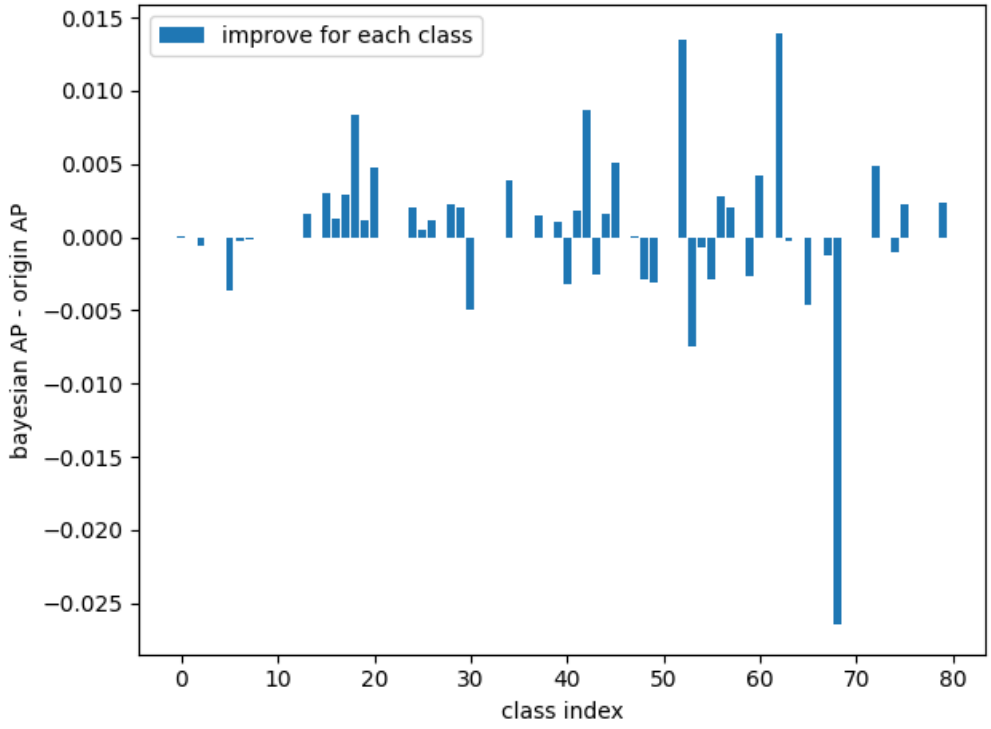

Figure 5.11 AP change for each class between Bayes v1 with origin

5.3.3 Bayes modifier v2

The second attempt are that only recalculate the probability of those ambiguous anchors. The mAP increased subtle continuously, from 0.5416 increased on 0.5418. By comparing each AP, the difference between v2 model with origin model was shown in figure 5.12

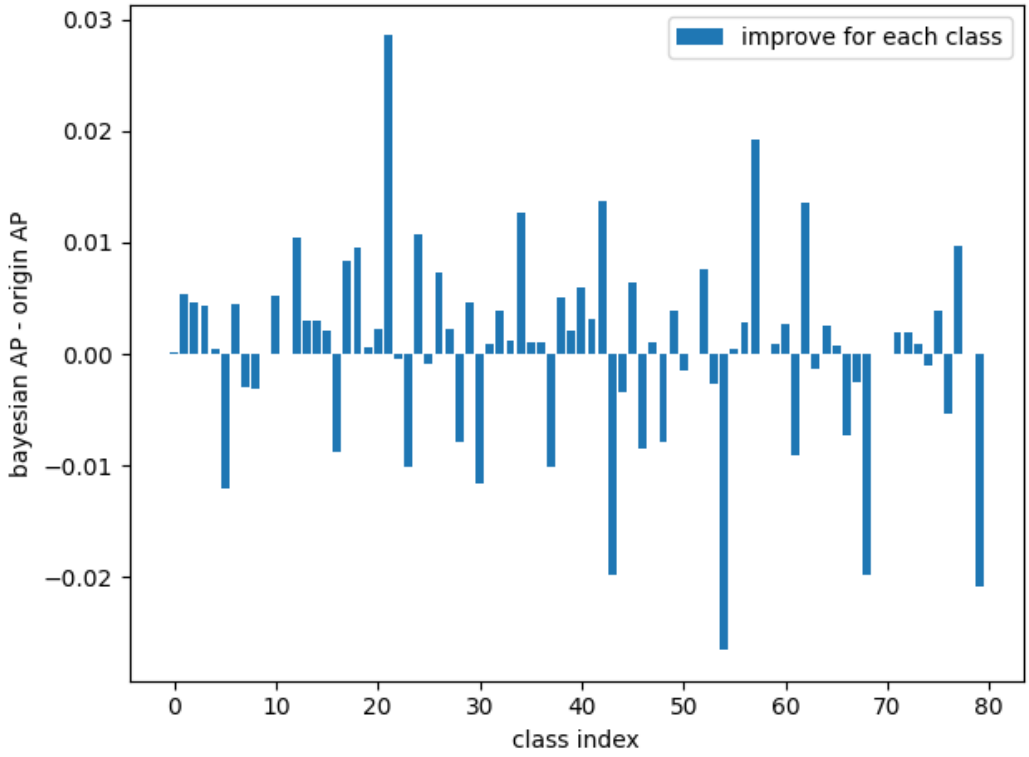

Figure 5.12 AP change for each class between Bayes v2 with origin

It is clearly that this version has better performance in improving the mAP of some classes.

5.3.4 Bayes modifier v3

In the former version, for bayes calculation, the prior probability is obtained based on the probability in COCO dataset. In this version, this prior probability has changed to the class score from YOLOv3 network. With setting the strong threshold as 0.9 (only those objects with confidence*class score > 0.9 can be chosen), and 0.4 ambiguity threshold (only those classes that (best class score - this class score) < 0.4 * best class score), the mAP improved from 0.5418 to 0.5420 as figure 5.13

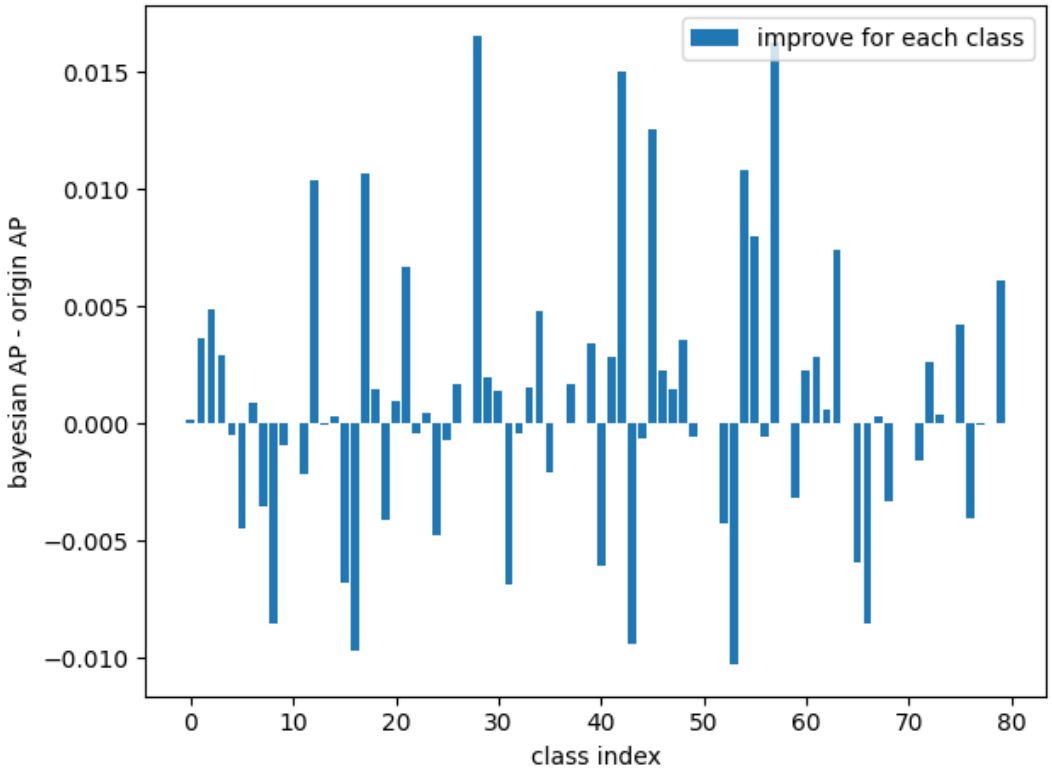

Figure 5.13 AP change for each class between Bayes v3 with origin

In addition, since the modifier considers more about YOLOv3 output, the ambiguity could be increased, from 0.4 to 0.5, and the mAP improved to 0.5423 , shown as figure 5.14

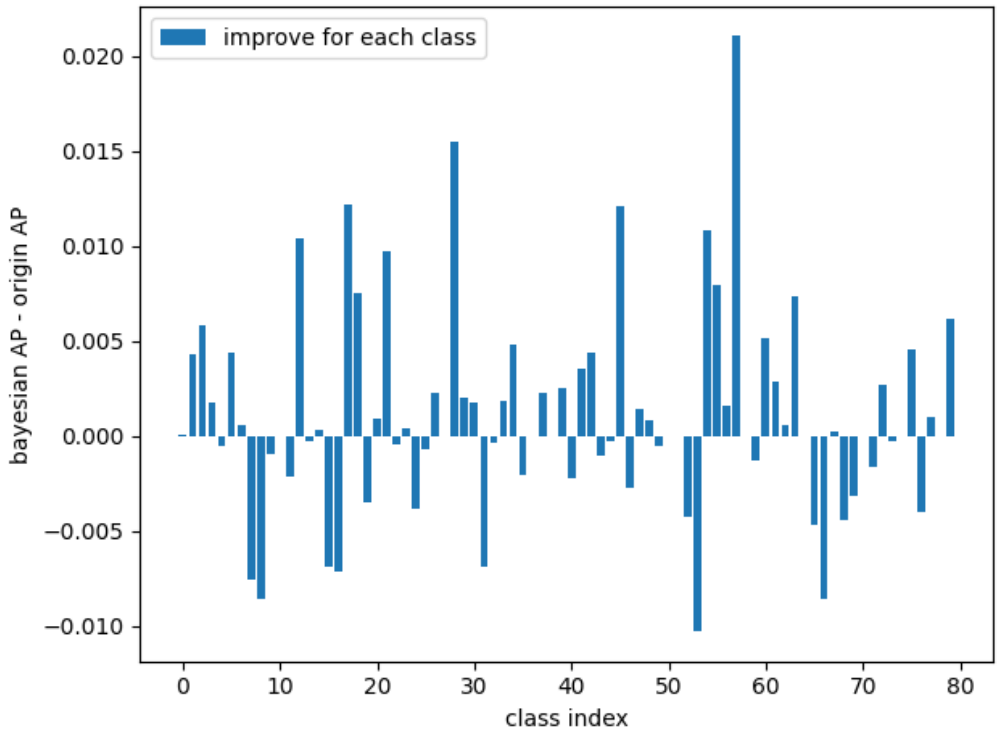

Figure 5.13 AP change for each class between Bayes v3 with origin

Chapter 6
Evaluation and Conclusion

6.1 Evaluation for project

Overall, this project aims to reproduce YOLOv3, then mainly improve the performance in light weight training, and make improvement based on trained weight from author. This project finally reproduced YOLOv3 through Torch, and made PASCAL VOC and COCO run well on it, although the 416*416 0.5mAP is 0.5412 which is little lower than the origin paper of 0.551. In addition, three different modifications were made on this basis:

1. The improvement of IoU to CIoU is considered to be a reproduction of the work of others. After completing the code part through the CIoU formula, the training experiment on the PASCAL VOC2007 data set has made the expected progress (the training result of 500 epoch has increased from 0.367mAP to 0.392mAP),

2. The principle of loss vector is relatively simple. By generating a weight vector to modify the loss function of YOLOv3, compared to adjusting the learning rate and only increasing the step size, the modification of loss better points out the direction. The effect of this improvement has also been higher than expected (0.367 mAP to 0.617 mAP).

3. The last proposed model innovation—Bayes modifier has undergone three iterations and multiple parameter adjustments in between, although each version performs better on COCO than the previous version (0.5412->0.5416->0.5418->0.5420- >0.5423), but the increment is not significant. It is worth mentioning that Bayes modifier should also be feasible in other target detection algorithms.

All in all, for the three improvements of YOLOv3, the performance is better on the main goal (improve performance in light weight training), and the performance on the secondary goal (improve accuracy based on weight from author) is average.

6.2 Further potential work

Since the method of bayes modifier only need the probability based on any dataset, and anchor confidence with class probability, it could be adapted in YOLOv4 and YOLOv5. Thereby, if this method can work well in other better object detection method, it can stand on the shoulders of giants to get a better than better accuracy.

Continuous improvement of the bayes modifier is also indispensable. Since the relative location of different object in the image should also be a quite important condition. If this information extracted properly, the model must have better performance.

MS COCO dataset is quite huge and common, which means it is mainly used to detect those state-of-art object detectors. However, there are also a lot of application scenarios in daily life. Hence, find an area that those improvements can have better performance is also a valuable direction.

Finally, due to the limitation of computing power, this project has not get sufficient in COCO or PASCAL VOC (although the aim does not include this). After the project, further training based on those change could be implemented on free virtual machine for months (more than 100 pound costed in cloud server renting).

6.3 Reflection

As a powerful object detection algorithm released in 2018, YOLOv3 contains a lot of detailed knowledge that I am unfamiliar with, and I have to use almost unfamiliar technology (torch) to understand its details and make improvements, which cost me about three months. Through searching for a large number of videos on YouTube and BiliBili for a month, and found many YOLO papers on IEEE, CVPR, and CNKI, I accomplish the mainly study part in the end of first semester.

In the learning process, the learning method that helped me the most is to follow the review paper to clarify the development of YOLO and target detection, as well as some key technologies. For unfamiliar bugs encountered in the code, the most helpful way is to use both the debug function of PyCharm and the issue section of GitHub.

Extremely difficult challenges have brought great gains while consuming a lot of energy. First of all, thanks to the research related literature, not only I have a deeper understanding of YOLOv1 to YOLOv3, but also a lot of related technologies in the field of target detection, which also motivates me to continue to engage in the field of computer vision in the future.

List of references

[1] Pathak, A., Pandey, M. and Rautaray, S., Application of Deep Learning for Object Detection, 2018.

[2] R. Girshick, “Fast R-CNN,” in Proc. ICCV, 2015, pp. 1440–1448

[3] Zhong-Qiu Zhao , Member, IEEE, Peng Zheng, Shou-Tao Xu, and Xindong Wu , Fellow, IEEE Object Detection With Deep Learning: A Review, 2019

[4] Y. LeCun et al., “Deep learning,” Nature, vol. 521, pp. 436–444, May 2015.

[5] Goodfellow I, Bengio Y, Courville A, et al. Deep learning (Vol. 1, No. 2) [J]. 2016.：326-366

[6] Gu J, Wang Z, Kuen J, et al. Recent advances in convolutional neural networks[J]. Pattern Recognition, 2018, 77: 354-377.

[7] A. Krizhevsky et al., “ImageNet classification with deep convolutional neural networks,” in Proc. NIPS, 2012, pp. 1097–1105

[8] R. Girshick et al., “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. CVPR, 2014, pp. 580–587.

[9] S. Ren et al., “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. NIPS, 2015, pp. 91–99.

[10] K. He et al., “Mask R-CNN,” in Proc. ICCV, 2017, pp. 2980–2988.

[11] K. He et al., “Spatial pyramid pooling in deep convolutional networks for visual recognition,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 9, pp. 1904–1916, Sep. 2015.

[12] T.-Y. Lin et al., “Feature pyramid networks for object detection,” in Proc. CVPR, 2017, pp. 936–944

[13] X. Ren and D. Ramanan, “Histograms of sparse codes for object detection,” in Proc. CVPR, 2013, pp. 3246–3253.

[14] J. R. R. Uijlings et al., “Selective search for object recognition,” Int. J. Comput. Vis., vol. 104, no. 2, pp. 154–171, Apr. 2013.

[15] Bochkovskiy A, Wang C Y, Liao H Y M. Yolov4: Optimal speed and accuracy of object detection[J]. arXiv preprint arXiv:2004.10934, 2020.

[16] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[17] J. Redmon and A. Farhadi. (2016). “YOLO9000: Better, faster, stronger.” [Online]. Available: https://arxiv.org/abs/1612.08242

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed.SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015. 5, 6

[19] Redmon J, Farhadi A. Yolov3: An incremental improvement [J]. arXiv preprint arXiv:1804.02767, 2018.

[20] CNN Explainer [Online]. Available: https://poloclub.github.io/cnn-explainer/

[21] Yu J, Jiang Y, Wang Z, et al. Unitbox: An advanced object detection network[C]//Proceedings of the 24th ACM international conference on Multimedia. 2016: 516-520.

[22] Rezatofighi H, Tsoi N, Gwak J Y, et al. Generalized intersection over union: A metric and a loss for bounding box regression[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 658-666.

[23] Zheng Z, Wang P, Liu W, et al. Distance-IoU loss: Faster and better learning for bounding box regression[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(07): 12993-13000.

[24] Erik Linder-Norén ,GitHub. 2021. eriklindernoren/PyTorch-YOLOv3. [online] Available at: <https://github.com/eriklindernoren/PyTorch-YOLOv3>

Appendix A
External Materials

Through the recommendation of my supervisor, the YOLOv3 model implemented by torch from GitHub (https://github.com/eriklindernoren/PyTorch-YOLOv3) has been referenced to reimplement YOLOv3.

Those basic python package are necessary

numpy

torch>=1.2

torchvision

matplotlib

tensorboard

terminaltables

pillow

tqdm

imgaug

PASCAL VOC2007 dataset has been used for training, which could be found through http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

COCO2014 dataset has also been used, which could be found through https://cocodataset.org/

Appendix B
Ethical Issues

Since all of the data were provided through common well-known dataset, no harm happened during the project, and no privacy has been stolen.

As an object detection, all of the categories were come from COCO and PASCAL VOC, thus there is no sensitive information could be extracted through the algorithm.

Appendix C
Source Code

The majory part of this project has been uploaded to GitHub:
https://github.com/Dan944/project

In the web page there is an README file which can introduce how to use it.

The whole project include network weight, downloaded dataset has been uploaded to
https://featurize.cn/