License Plate Detection Using YOLO

Yikun Wu, Chenchen Mao, Yingjie Wu, Keyang Li, Jiaxin Song

May 2020

Abstract

This report focuses on using YOLOv3 to detect license plates. It gives a short background on YOLOv3 and explains its network architecture, learning and testing process and memory computation. It also showcases the retaining and demo parts of detecting plates. YOLOv3 is more powerful than previous versions since its network structure is more complex. Also, new techniques are added to address problems such as detecting small objects while maintaining high speed. Our team studied and analyzed source code and papers about YOLOv3, and applied this powerful model to license plate detection. This report demonstrates the research and work we have done in detail.

Contents

Chapter 1

Introduction

Licence plate detection with deep neural network can be applied widely worldwide. The ability to detect vehicle license plates can contribute to a safe environment for driving. As this project can help to detect vehicles which exceed speed limit on the road. You Only Look Once(Yolo) is a popular object detection model. Our project focuses on detecting license plate with YOLOv3.

1.1 Understanding YOLOv3

YOLOv3 takes images with license plate as input. Then it resizes images into 416x416x3 and divides images into SxS grid. Each grid predicts B number of bounding boxes. Each bounding box has x,y,w,h,and confidence(object)

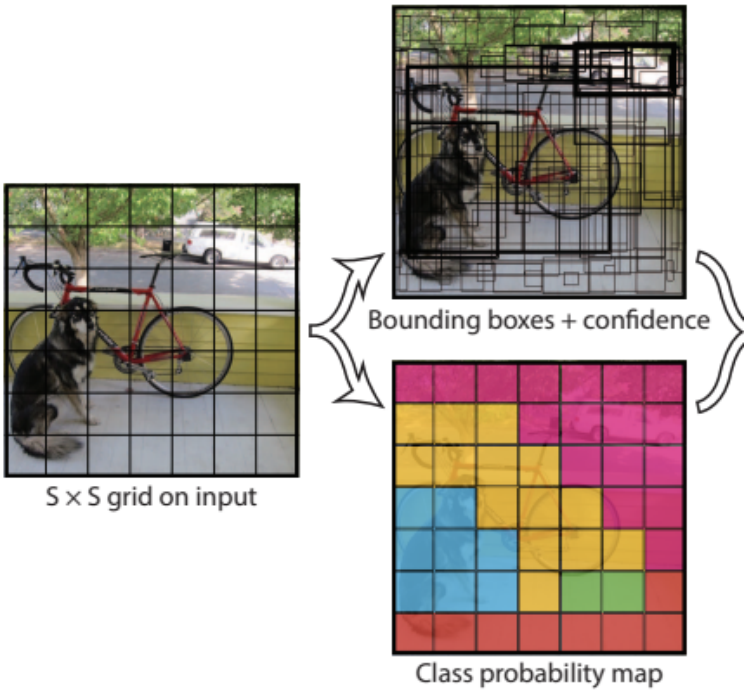

Figure 1.1: YOLOv3

Each bounding box also predicts probabilities of C number of categories. YOLOv3 generates three feature maps that are $13 * 13 + 26 * 26 + 52 * 52$. Therefore, the maximum number of bounding boxes potentials generated by YOLOv3 is $(13*13+26*26+52*52)*3 = 10647$ while YOLO V2 gernates 13x13x5=485 bounding boxes. This is why YOLOv3 is more powerful to detect small objects.

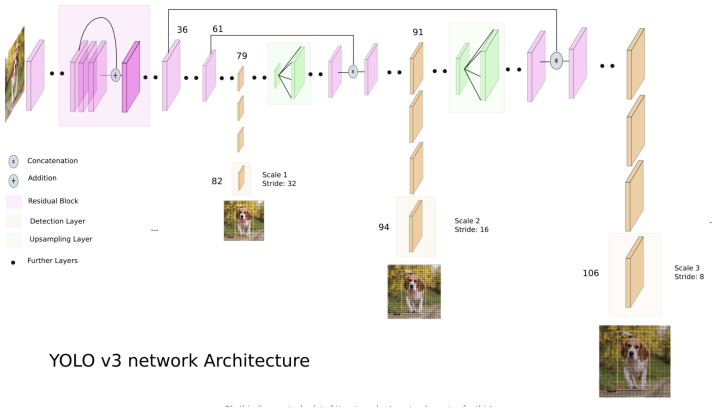

Figure 1.2: Feature Map of YOLOv3

In those bounding boxes, some of them are overlapping bounding boxes. In order to eliminate overlapping bounding boxes, YOLOv3 uses Non-maximal suppression to fix multi detection problem. For instance, YOLOv3 discards all bounding boxes with prediction lower than 0.6. Then YOLOv3 chooses box A with largest prediction and discard all the box has IoU with box A ≥ 0.5. After all the steps above, YOLOv3 produce images with bounding boxes and labels.

1.2 Detecting License Plate

In order to train YOLOv3, we manually labeled hundreds of images with license plate. At same time, we used pre-trained model online. In order to detecting license plate, there are several modifications in YOLOv3. The license plate YOLOv3 resizes image into 608x608x3 instead of 416x416x3. Also,the feature maps are $19 * 19 + 38 * 38 + 76 * 76$. Therefore, license plate YOLOv3 generates $(19 * 19 + 38 * 38 + 76 * 76) * 3 = 22743$ bounding boxes, which is more the number of bound.

Chapter 2

Network Architecture

In terms of basic image feature extraction, YOLOv3 uses a network structure called Darknet-53 (contains 53 convolution layers), which draws on the practice of residual network and sets up shortcut links between some layers (Shortcut connections).

2.1 Network Structure

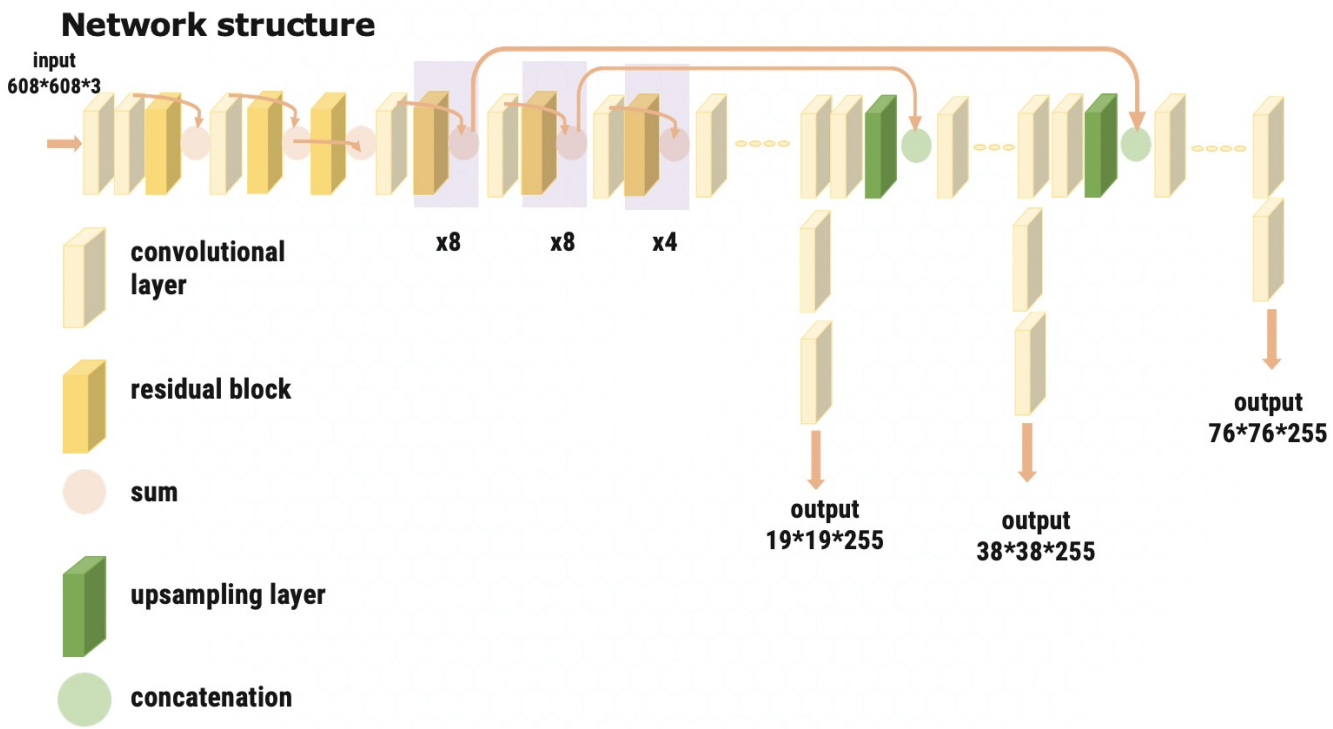

Figure 2.1: Network Structure

Figure 2.1 shows the basic network structure of YOLOv3. There are convolution layers, residual blocks and upsampling layers. Sum and concatenation are used to synthesize information after certain processing.

2.1.1 Convolution layer

convolution layers are used to fetch the features of images. Multiple convolution stacking is to continuously score images and feature maps. The deeper the network, the stronger the abstraction ability, but not the deeper the better. A network that is too deep will cause the gradient to disappear or the gradient to explode. In order to solve these problems, the Batch

Norm layer was introduced. The essence of the Batch Norm layer is to use changes to change the size of the variance and the mean position, so that the data is more in line with the distribution of real data, ensuring the nonlinear expression of the model.

2.1.2 Residual block

Residual block is something new in YOLOv3. Neural networks are known as universal function approximators and that the accuracy increases with the increasing number of layers. However, if we still keep increasing the number of layers, we will see that the accuracy will start to saturate at one point and eventually degrade. It might seem that the shallower networks are learning better than their deeper counterparts. But this is what is seen in practice and is popularly known as the degradation problem.

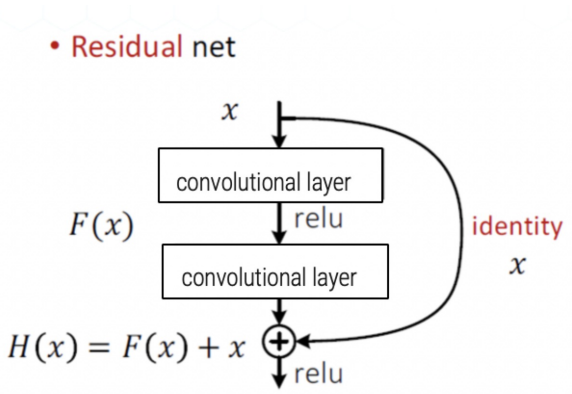

Figure 2.2: Residual Block of YOLOv3

In the traditional convolution layer or fully connected layer, there are more or less information loss problems when information is transmitted. Residual block solves this problem to some extent. By directly bypassing the input information to the output to protect the integrity of the information, the entire network only needs to learn the part of the difference between input and output, simplifying the learning goals and difficulty. The residual layer here is made of two convolution layers as we can see in Figure 2.2.

We can know that ResNet can be simply expressed as follows

$$x_{l+1} = x_l + F(x_l, W_l)$$

Where $X_l + 1/X_l$ respectively represent the output and input of the l th layer, then for the unconnected L th layer and l th layer, the corresponding learned features can be expressed as

$$x_L = x_{L-1} + F(x_{L-1}, W_{L-1}) = x_{L-1} + x_{L-2} + F(x_{L-2}, W_{L-2}) = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

Next, in the process of derivation of the reverse gradient, when the final loss is derivated at X_l, it can be obtained

$$\frac{\partial loss}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial x_L}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot \frac{\partial (x_l + \sum_{i=l}^{L-1} F(x_i, W_i))}{\partial x_l} = \frac{\partial loss}{\partial x_L} \cdot (1 + \ \ldots)$$

We can see that the derived derivative is the derivative of the L layer multiplied by the number in parentheses. There is a 1 indicating that the gradient of the L layer can be propagated directly back without loss.

2.2 Output

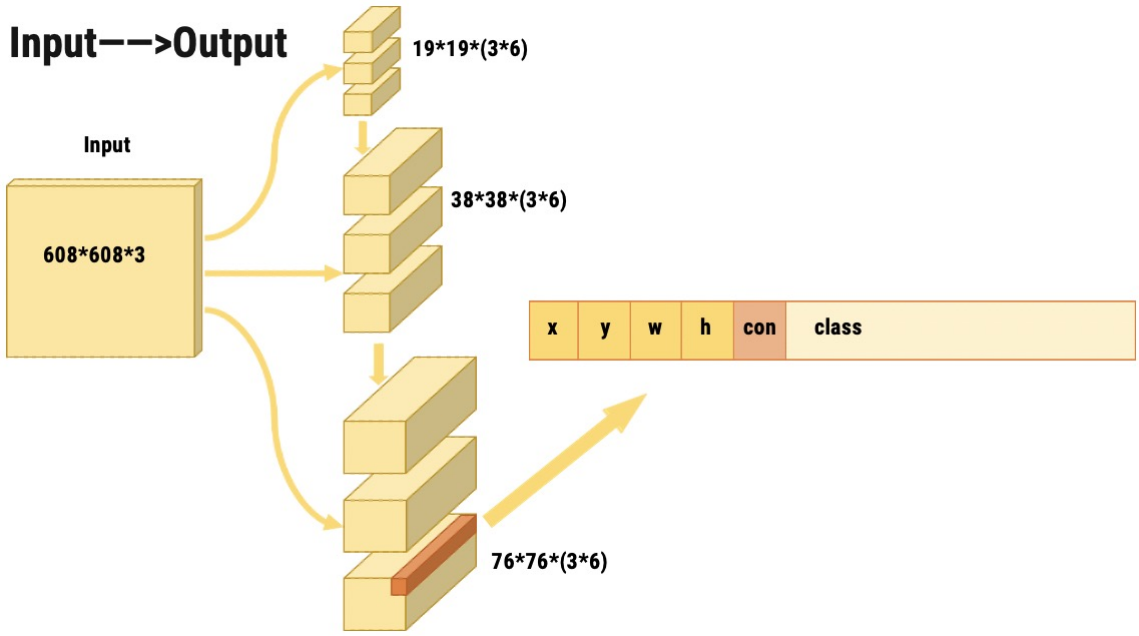

Figure 2.3: Output of YOLOv3

As is shown in Firgure 2.3, the input of YOLOv3 is a 608x608x3 tensor and the outputs are three tensors with different sizes.

The third dimension of each output tensor which is 3x6 means that each grid cell has three kinds of anchor boxes and each anchor and six relative values. The total anchor boxes here can be calculated as:

19x19x3+38x38x3+76x76x3=**22743** anchor boxes

This means we have the same amounts of predictions.

Each prediction is a 6-dimensional vector. X and y represent the position of the center of the anchor box. W and h represent the size of the anchor box. And there is also value about confidence score. The last value is the accuracy of each class. Since we only detect plates, there is only one class accuracy value.

We also have a YOLO layer to decode information and use them to calculate prediction bounding box variance and class variance.

Chapter 3

Learning and Testing

In the previous chapter, we learned that YOLOv3 adopted new techniques such as residual blocks, upsampling and skip connections. In this chapter, we will further examine how YOLOv3 improves accuracy for small object detection, classifies objects, calculates loss, and selects priors. Meanwhile, we will discuss about paramaters in the configuration file, and how learning rate changes during the learning process.

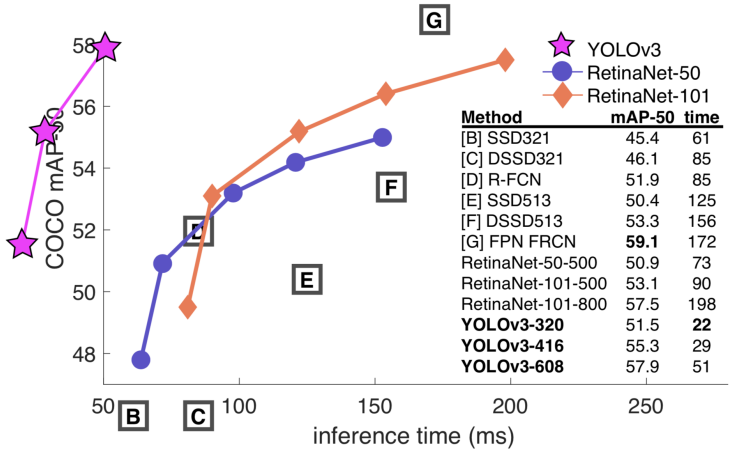

Figure 3.1: **Performance of YOLOv3.** Speed/accuracy tradeoff on mAP with IOU = 0.5. Detection is done at three different scales. Average precision(AP) or mean average precision(mAP) is the metrics commonly used for comparing various object detectors.

3.1 Small Object Detection

One issue that YOLOv2 struggled most is to detect small objects accurately. In YOLOv2, the input is downsampled through multiple layers, such as pooling layers, in the network. After downsampling, some fine-grained information are lost. This leads to unsatisfying performance in detecting small objects.

YOLOv3 targets this drawback with new approaches. First of all, it adopts a stronger, deeper and more complex underlying architecture, the Darknet-53. Detections are done with a 1x1 kernel at different sized feature maps at different places in the network. The model downsamples input dimensions by 32, 16 and 8 respectively, resulting in three feature maps. A 13x13 layer is used for detecting large objects, a 52x52 layer for detecting small objects and a 26x26 layer for detecting medium objects. By making predictions at three different scales, this model can keep some semantic information from upsampled features while not losing fine grained information from earlier layers.

3.2 Class Prediction

YOLO uses a softmax function to calculate class probabilities for each label and label the class with the highest probability. The softmax is based on a mutually-exclusive assumption; thereby, an object is either labelled as a person or a child.

However, YOLOv3 uses independent logistic classifiers to replace the softmax function. A multi-label classification method is used. It is based on non-mutually exclusive assumption, and therefore overlapping labels is possible. Hence, an object may be labelled as both a person and a child. This change also helps to reduce computational complexity.

3.3 Loss Function

Yolo calculates Loss Function using the traditional method Mean Squared Error. With MSE, regression is done on the center point coordinates (x, y) and height (h) and width (w) of bounding box.

However, YOLOv3 uses binary cross-entropy loss instead of mean square error.

3.4 Anchor Box Selection

Anchor box, also named priors, is used during object detection. One change YOLOv3 adopts is using an advanced algorithm called k means clustering to generate anchor boxes. Previously, YOLO uses hand-picking methods to generate anchor boxes. As an efforts to optimize performance, YOLOv3 uses k mean clustering method so that sizes of anchor boxes are generated through a learning process that customize the objects and help to improve test accuracy. In V3, 9 anchors boxes are implemented with 3 for each scale.

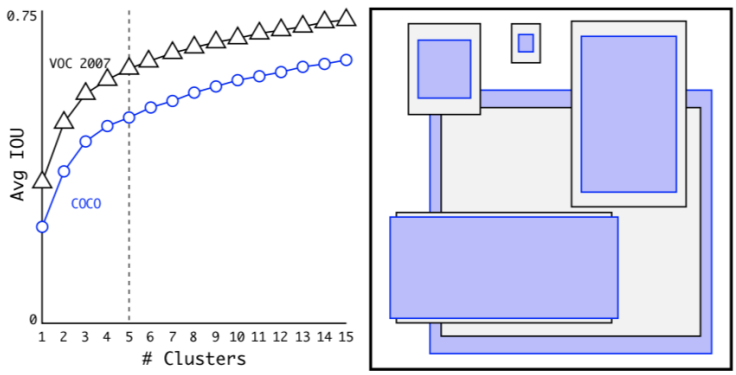

Figure 3.2: k means clustering. When k = 5, five priors are generated.

3.5 Parameters

In this section, we will talk about some parameters in the configuration file of YOLOv3. For testing, set batch = 1 and subdivisions = 1.

```
batch = 64
subdivisions = 16
width = 416
height = 416
channels = 3
momentum = 0.9
decay = 0.0005
```

batch. Batch size indicates size of the subset of imaged used in one iteration for training.

subdivisions. Subdivisions helps to process a fraction of batch such that the GPU is able to handle the batch with its memory. GPU will process batch/subdivisions number of images at a time.

weight, height, channel. Input image has 3 channels (RGB), and will be resize to width x height prior to training.

momentum. Because weights are updated with a batch-sized number of images, changes in weights can varies. Thus, momentum is used for penalizing big weight changes across iterations.

decay. Overfitting means the model is performing well on training data but poorly on test data or real world application datasets. To avoid overfitting, decay is for controlling the penalty for large weights.

3.6 Learning Rate

```
learn_rate = 0.001
burn_in = 1000
max_batches = 50200
policy = steps
steps = 40000, 45000
scales = .1, .1
```

learning-rate. The parameter learning-rate indicates how aggressively the model learns with current batch of data. Often times, learning-rate is set to be high at the beginning, decreases over time, and ranges in [0.01, 0.0001].

policy, steps. Decreases in learning rate is specified by setting policy to steps. After tons of iterations, new learning rate will be updated through scaling with the parameter scales.

burn-in. The parameter burn-in controlls a short period of time when we have a lower learning rate at the beginning. This may help to speed up the training to some extend.

In other words, the learning rate will change as the number of learning times grows. The parameter policy defines the policy for adjusting the learning rate when learning times is bigger than burn-in. When the number of iteration is less than parameter burn-in, learning rate is calculated as below:

```
if batch_num < burn_in:
    return learning_rate * pow((float)batch_num / burn_in, power)
```

Chapter 4

Memory and Computation

Compare to predecessors in object detection such as R-CNN and Fast R-CNN, YOLO has significant speed advantage. This speed advantage is largely due to its network architecture. This chapter will analyze the computation complexity from the perspective of the network architecture. The memory usage will also be discussed in this chapter.

4.1 Computation Complexity

As the previous chapter show, the whole network is mainly consist of convolution layer. In that case, most of the computation is generated in convolution layer. When a feature map go through a convolution layer, pixel value will multiply the value in corresponding position of the kernel matrix and then add them all with bias as the new pixel value of the new generated feature map. Then the kernel will slide to other area and repeat the same operations.

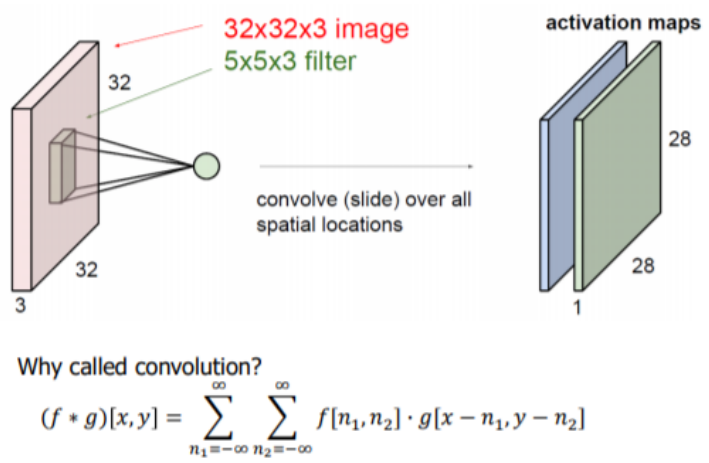

Figure 4.1: Examples of how the convolution layer work(activation map is also called feature map)

In the process mentioned above, there are two basic operations involved. For each convolution layer, The format of calculating the number of addition and multiplication is $W \times D \times C \times F \times S \times S \times 2$, where $W \times D$ is the size of one input feature map, C is the number of channel, F is the number of filters, and $S \times S$ is the size of kernel.

Besides the addition operations in convolution layer, there are also some addition operations happen in the output of the residual block. The input of the residual block and the output of the residual block will be added and then forward to next layer. In this process, the number of addition operations in each residual block can be calculated by $W \times D \times C$.

According to the analysis of the source, only the last three convolution layers contains bias.

The total number of operations in one forward propagation is $1.40197E + 11$.

4.2 Memory Requirement

Basically, the memory usage of object detection network consist of two parts, the memory used to store parameters such as weights and bias and the memory used to store generated feature maps during the forward propagation and backward propagation. Parameters can be divided into two trainable parameters such as weights and bias, and non-trainable parameter such as some unchanged parameters in the batch normalization layers. In YOLOv3, they are stored as float type, which cost 4 bytes. Feature maps are matrix storing float numbers. The total memory usage is the combination of these two parts.

4.2.1 Parameters

In YOLOv3, weights are float numbers in kernel matrix, and bias are the same per filter. In that case, the trainable weights in each convolution layer can be calculated by $S \times S \times F$, where $S \times S$ is the shape of filter in each layer and F is the number of filter in each layer. The number of bias, which only used in last three output layer, is 3 $timesF$, where F is the filter number in last three output layer.

In addition, there are also some trainable parameters in the batch normalization layer following each convolution layer. The equations in Figure 4.2 shows what happen in batch normalization layer. In this two equations, gamma and beta are trainable, while variance and mean are non-trainable.

$$w_{new} = \frac{gamma * w}{sqrt(variance)}$$

$$b_{new} = \frac{gamma*(b - mean)}{sqrt(variance)} + beta$$

Figure 4.2: Operations in batch normalization layer

The total number of trainable parameters and non-trainable parameters in YOLOv3 are 61584406 and 52608 respectively.

4.2.2 Feature Map

The generated feature maps are output matrices of each convolution layer. We can calculated the number of float numbers of matrices after each convolution layer by $W \times D \times F$, where $W \times D$ is the shape of the generated feature map, and F is the number of filters in this layer. The size of memory required by feature maps is also affected by the batch size. Batch size is the number of initial images in one iteration. By learning the configuration file of YOLOv3, we find that during the training process, the batch size is usually set 64, which means 64 images will be loaded into memory in one iteration. These 64 images are divided into 16 mini-batch, so the YOLOv3 network will calculate 4 images in one mini-batch. The memory required by

feature maps will vary as the size of batch and mini-batch. The total memory usage is 440M when the batch size is 64 and mini-batch is 4.

4.2.3 Total Memory Usage

According to the configuration file of YOLOv3, during the backward propagation, YOLOv3 implement momentum during the backward propagation, which means the computer needs the same size of memory required by trainable parameters for gradients and momentum in backward propagation. Differ from feature map, YOLOv3 will calculate the loss of all images in one batch and sum them together before the start of backward propagation, which means the memory required by parameters will not vary as the size of batch. The composition of the total memory usage is shown in figure 4.3. The calculated result is 4.3G, which is very closed to the real memory usage during the training process.

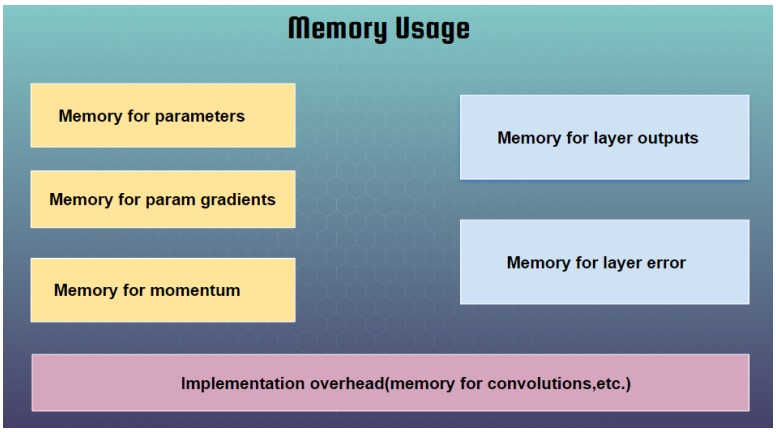

Figure 4.3: The composition of memory usage

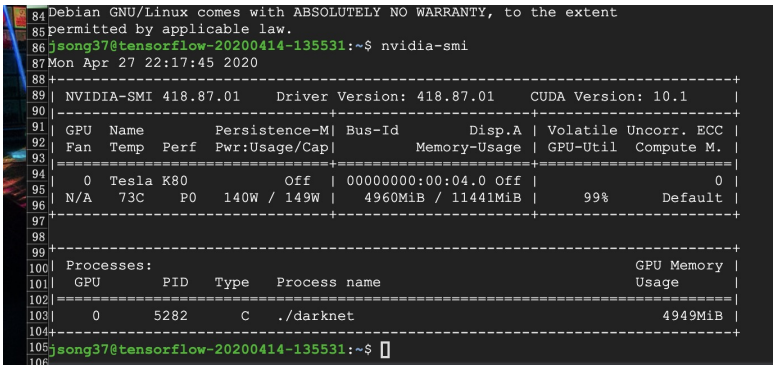

Figure 4.4: The real memory usage

Chapter 5

Retrain and Demo on VOC Dataset

5.1 Training on VOC

This part of job is Training YOLOv3 model on VOC data set.

At first, We downloaded Pascal VOC Data and make a directory to store it. Then We generated the label files that Darknet 53 uses. Darknet wants a .txt file for each image Where x, y, width, and height are relative to the image's width and height. For training we use convolution weights that are pre-trained on Imagenet.

The result is shown in the Fig5.2. The model spend about points 1 second to predict one image.

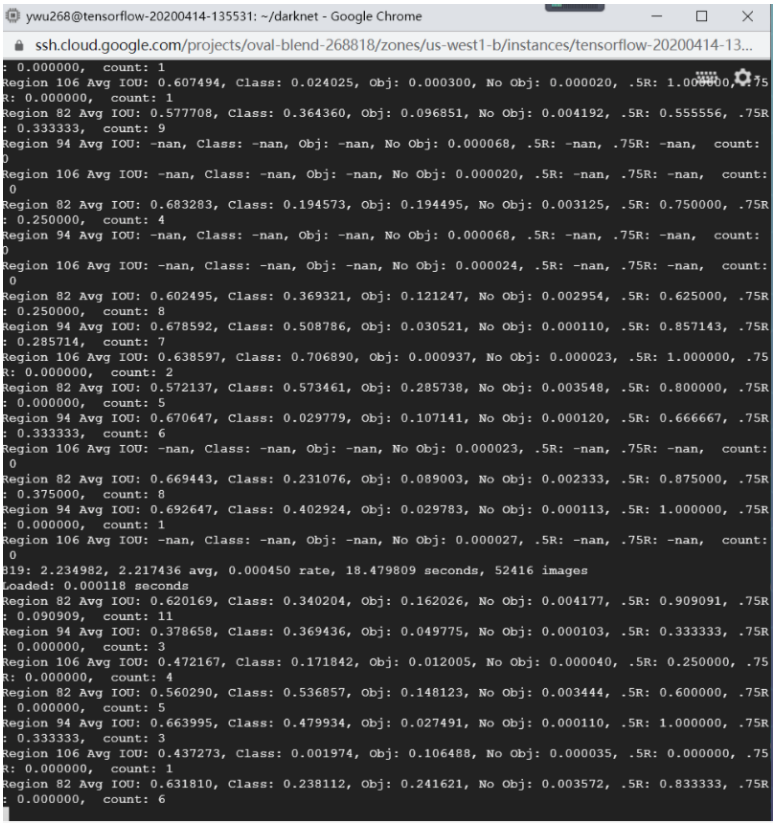

Figure 5.1: Training

```
  62 conv   1024  3 x 3 / 2    26 x  26 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  63 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  64 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  65 res   62                  13 x  13 x1024   ->    13 x  13 x1024
  66 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  67 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  68 res   65                  13 x  13 x1024   ->    13 x  13 x1024
  69 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  70 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  71 res   68                  13 x  13 x1024   ->    13 x  13 x1024
  72 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  73 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  74 res   71                  13 x  13 x1024   ->    13 x  13 x1024
  75 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  76 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  77 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  78 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  79 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512  0.177 BFLOPs
  80 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
  81 conv    255  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 255  0.088 BFLOPs
  82 yolo
  83 route  79
  84 conv    256  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 256  0.044 BFLOPs
  85 upsample            2x    13 x  13 x 256   ->    26 x  26 x 256
  86 route  85 61
  87 conv    256  1 x 1 / 1    26 x  26 x 768   ->    26 x  26 x 256  0.266 BFLOPs
  88 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512  1.595 BFLOPs
  89 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256  0.177 BFLOPs
  90 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512  1.595 BFLOPs
  91 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256  0.177 BFLOPs
  92 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512  1.595 BFLOPs
  93 conv    255  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 255  0.177 BFLOPs
  94 yolo
  95 route  91
  96 conv    128  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 128  0.044 BFLOPs
  97 upsample            2x    26 x  26 x 128   ->    52 x  52 x 128
  98 route  97 36
  99 conv    128  1 x 1 / 1    52 x  52 x 384   ->    52 x  52 x 128  0.266 BFLOPs
 100 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256  1.595 BFLOPs
 101 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128  0.177 BFLOPs
 102 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256  1.595 BFLOPs
 103 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128  0.177 BFLOPs
 104 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256  1.595 BFLOPs
 105 conv    255  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 255  0.353 BFLOPs
 106 yolo
Loading weights from VOCbackup/yolov3-voc.backup...Done!
VOCdevkit/VOC2007/JPEGImages/000275.jpg: Predicted in 0.116856 seconds.
person: 98%
horse: 95%
```

Figure 5.2: Prediction

There are several predict boxes to show the object detected by the model.

Figure 5.3: Output

Chapter 6

License Plate Detection

YOLOv3 allows users to modify the configuration files and to apply the model on their own data set. In this project we apply YOLOv3 on license plate detection.

6.1 Preprocessing of Training Set

In order to train the model on our license plate data set, for each image in data set we need to generate XML file which contains the objects positions and category. We use a software called LabelImg to label images and generate required XML files (Figure 6.1)

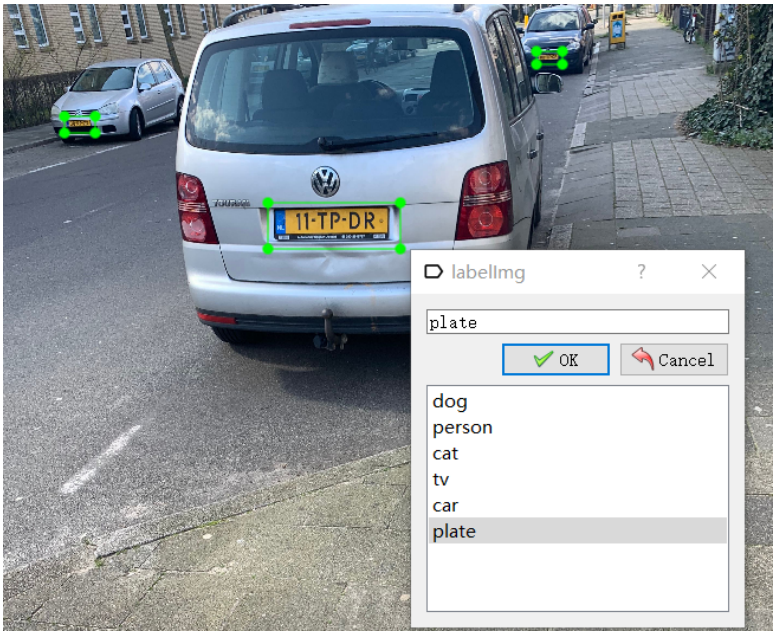

Figure 6.1: Label images

These XML files are then input into a python script called voc label to generate the TXT files containing the position and category information of each bounding box in each image. YOLOv3 will calculate loss according to the information in these TXT files.

6.2 Configuration

There are three configuration files need to be modified. The voc.names file contains all the classes name in data set, for our license plate data set we only need to write plate into this file. The voc.data file contains the names and paths of your training and validation set. It also tells YOLO which is the .names file for current data set and where to store the generated weights file. The YOLOv3.cfg file is the file containing the configuration parameters of the network. This

file control parameters such as batch size, learning rate, learning rate update policy, momentum and so on. It also defines the kernel size, number of filters, slide in each convolution layer. In order to apply YOLOv3 on license plate data set, we change the number of classes in this file to 1, what's more, the number of filters in last three output layers should vary as the number of classes. The number of filters in these layers is calculated by $3 \times (5 + number of classes)$, where 3 is the number of predicted boxes, 5 is five value for the position information and confidence. In license plate data set, the number of filter is 18(see Figure 6.2).

We also modified the darknet.c file. This file provides commands to control YOLOv3. In this file, the default .names file is changed into our modified .names file.

```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 6.2: YOLOv3.cfg file

6.3 Training and Testing

The size of the license plate training set is 237. We trained the model for 1600 times. The loss after 1600 times train is 0.1.

Figure 6.3: Example of the result of trained model

We then calculate the mAP of our data set. mAP is mean Average Precision. It computes the average precision val ue for recall value over 0 to 1. (Precision measures how accurate is your predictions and Recall measures how good you find all the positives)

There are two test sets, one set only contains Indian car plates, the other one contains plates in different countries and different type of vehicles such as motorbike.

The performance on each test is different. The size of the set only containing Indian car plates is 37 and the mAP is 0.78(Fig 6.4). The size of the set containing plates from different countries and vehicles is 64 and the mAP is 0.48 (Fig 6.5). As a comparison, the mAP of VOC data set trained by YOLOv3's developers is 0.77.

```
Reading annotation for 1/39
Saving cached annotations to ./annots.pkl
plate_ap:0.7816883591554644
cache file:annots.pkl has been removed!
map:0.7816883591554644
```

Figure 6.4: mAP of Indian license plate test set

```
Reading annotation for 1/64
Saving cached annotations to ./annots.pkl
plate_ap:0.4829218404896528
cache file:annots.pkl has been removed!
map:0.4829218404896528
```

Figure 6.5: mAP of license plates in different countries test set

6.4 Performance Analysis

The performance of the trained model on the set containing plate in different countries and different vehicles is not as high as it on the test set only containing Indian car plates. There are several reasons behind this result.

First of all, the size of training set is not bigger enough. There are only 237 images in the data set, which is not enough to provide the model to learn more information about license plates.

Second, the images in the training set only containing Indian car license plates. The license plates in different countries is different and they also vary as the type of vehicles. The test set contains license plate from different countries and different type of vehicles such as motorbike. It seems the trained model has a bad performance on the plates in different countries and different vehicle types, which reduce the mAP. This may be caused by overfitting. The model has learned too much about the features in Indian car plates so that it fail to detect some plates in different countries and vehicles.


(a) Indian car

(b) Other country's car


(c) Motorbike

Figure 6.6: Test set

Chapter 7

Summary

To summarize, the goal of this project is to learn about YOLOv3 and apply it to license plate detection. From this project, we learned about the network architecture, including the functionality of convolution layer, residual block, and upsampling layer, and how these parts work together to make this architecture stronger. Also, we learned about the inference function of YOLOv3, how it predicts objects based on the feature maps generated by network and how it uses techniques such as multi-label classification, logistic regression, binary cross-entropy loss, k-mean clustering to update network parameters, improve performance, detect small objects and adjust hyper parameters like learning rate. Moreover, we analyzed the memory requirement and computation complexity of the network, how to compute the number of trainable parameters and non-trainable parameters, how to compute the memory usage of feature maps and how to compute the number of required operations during the using of YOLOv3.

To implement YOLOv3, firstly we retrain the model on the VOC data set provided by the developers and figured out how to use it. Then we collected our own images and made them into data set that could be trained on YOLOv3 model. The trained model performs well on detecting Indian car plates and has limited accuracy when test data becoming more diverse. We believe that with more diverse training sets, this problem could be tackled.