Learning to Detect and Recognize Road Signs

Author: Puzhuo LIU (22089678)

Supervisor: Professor Yukun LAI

School of Computer Science and Informatics

Cardiff University

September 2023

Abstract

As autonomous driving technology continues to develop, accurate detection and recognition of road signs is crucial to achieving a safe and efficient autonomous driving system. This paper proposes a method of integrating attention mechanism based on YOLOv5 model for the task of road sign detection and recognition in automatic driving system, and applies data enhancement and dhash deduplication technology to optimize the training set.

First, I introduced an attention mechanism into the YOLOv5 model, including SE (Squeeze-and-Exitation) and CA (Cooperative Attention) mechanisms, to enhance the model's attention to landmark features. Secondly, to improve the generalization ability of the model, I introduced data augmentation technology. In addition, I also judge the similarity of the image by calculating the dhash value of the image, and remove the images with high similarity in the training set to avoid over-fitting of the model.

Experimental results show that the YOLOv5 model that introduces the attention mechanism has achieved effective performance improvement in the road sign detection task. At the same time, data enhancement and dhash deduplication technologies also effectively improve the training effect of the model, enabling it to achieve stable and excellent performance in different scenarios

Acknowledgements

Here I would like to thank my supervisor Professor Lai Yukun. He was careful and responsible throughout the entire project and provided me with many good opinions and suggestions. I would also like to thank my family for their constant support.

Table of contents

1 Introduction 1

2 Background 3

3 Optimization 14

4 Project implementation 22

5 Compare and test 52

6 Conclusion 56

7 Reflection on learning 57

1 Introduction

Today, the emergence of autonomous driving technology has become a major advancement in the development of human society. Autonomous driving can bring enormous value to drivers, the automotive industry, and society. Autonomous driving systems can make driving safer, more convenient, and more enjoyable. Time that used to be spent driving on the road can be used to video call with friends, watch a fun movie, or even work. For workers with long commutes, driving a self-driving car could increase employee productivity and even shorten the workday. As workers can get their jobs done in self-driving cars, they can leave the office more easily, which in turn could attract more people to rural areas and suburbs. Autonomous driving could also improve the mobility of older drivers, giving them alternatives to public transportation or car-sharing services. Safety is also likely to improve, with a study showing that the increasing adoption of advanced driver assistance systems (ADAS) in Europe could reduce the number of accidents by around 15% by 2030[1].

Image detection and recognition technology and artificial intelligence have developed rapidly in recent years, and have an important influence in the field of automatic driving. It provides the ability for automatic driving systems to perceive and understand the environment, thereby realizing intelligent decision-making and control of vehicles. This paper mainly studies and discusses the application of the recognition and detection functions of CV technology in the field of automatic driving, and realizes the recognition of road signs by artificial intelligence.

YOLOV5 is a computer vision model based on the Pytorch framework released by Ultralytics in 2020 used for detecting objects. YOLOv5 is designed to improve the speed and accuracy of object detection while maintaining relatively low computing resource requirements. Unlike traditional two-stage object detectors (such as Faster R-CNN), YOLOv5 adopts a single-stage detection method, which directly predicts the category and bounding box of the object in a single step. Yolov5 regards the target detection task as a regression problem, and directly predicts the category and bounding box information of the target through the convolutional neural network. It divides the image into a grid, and each grid cell is responsible for predicting whether an object is contained, as well as the object's category and bounding box information. This design enables YOLOv5 to strike a balance between real-time performance and accuracy, and is suitable for many computer vision applications, such as object detection, image recognition, and pedestrian recognition [2].

This project also discusses the performance improvement of the attention mechanism Coordinate Attention and Squeeze-and-Excitation Networks on the training set of YOLOV5 in complex environments such as rainy days, nights, and high light conditions with high noise. Use data augmentation methods to increase the diversity and complexity of datasets, thereby improving the robustness and generalization capabilities of recognition task models. At the same time, a dhash is used to the re-algorithm preprocesses the training set to improve the training effect. In this project, I used the Chinese traffic dataset, which contains two training sets for road sign recognition and detection.

2 Background

2.1 A More Comprehensive Traffic Sign Detection Benchmark

Traffic signs play a pivotal role in guiding vehicles, and the detection of these signs stands as a critical component within the realms of autonomous driving and intelligent transportation systems. The construction of a comprehensive traffic sign dataset, enriched with diverse samples encompassing an extensive range of attributes, signifies a significant stride in advancing research dedicated to traffic sign detection. In this vein, Jianming Zhang's research team introduced an innovative Chinese traffic sign detection benchmark. This benchmark builds upon the foundation of CCTSDB 2017[3] by augmenting it with over 4,000 authentic traffic scene images and meticulous annotations. Furthermore, it has thoughtfully replaced numerous previously straightforward images with more intricate samples to adapt to the complexities of dynamic detection environments. By amplifying the inclusion of challenging samples, this updated benchmark serves to enhance the overall resilience of detection networks, compared to its predecessor [4].

Additionally, the team has introduced a dedicated test set, thoughtfully categorized according to three key aspects: semantic meanings, symbol sizes, and varying weather conditions. To culminate this endeavor, nine well-established traffic sign detection algorithms underwent a comprehensive evaluation against this fresh benchmark. The benchmark proposed by Zhang's team serves as a compass, guiding future research endeavors within the algorithmic landscape, ultimately fostering the development of more accurate, robust, and real-time traffic sign detection algorithms [5].

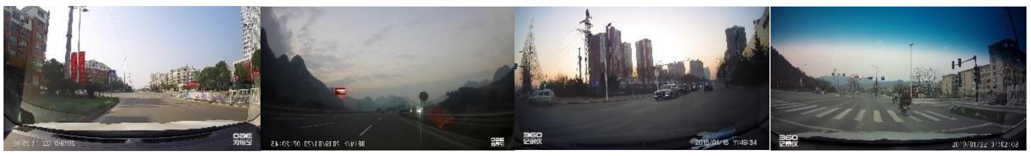


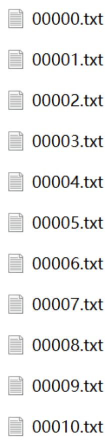
Figure 1 Example of training pictures from CCTSDB

In the cctsdb 2021 dataset, there are a total of 17856 images in the training set and positive sample test set. The categories of traffic signs in the image are divided into mandatory, prohibited and warning according to their meaning.

There are 16356 training set images numbered 00000-18991 in total.

The positive sample test set has 1500 images numbered 18992-20491.

The label format is txt, which can be directly trained by yolo. The information contained in each txt file is: category, bounding box x center coordinate, bounding box center y coordinate, bounding box width, bounding box height. All are normalized.



00000.txt
00001.txt
00002.txt
00003.txt
00004.txt
00005.txt
00006.txt
00007.txt
00008.txt
00009.txt
00010.txt

Figure 2 Example of training labels from CCTSDB

2.2 Chinese Traffic Sign Database

TSRD is the target recognition task training dataset for this project, it includes 6164 traffic sign images, which contain 58 sign categories. The images are divided into two sub-databases: a training database and a testing database. The training database contains 4170 images and the testing database contains 1994 images. All images are annotated with four coordinates of sign and category [6]. The difference between the recognition task and the detection task is that the detection objects in the recognition task training set will account for more than 80% of the overall image, and there will be no influence of light or environment. But this data set will have more classes than cctsdb.

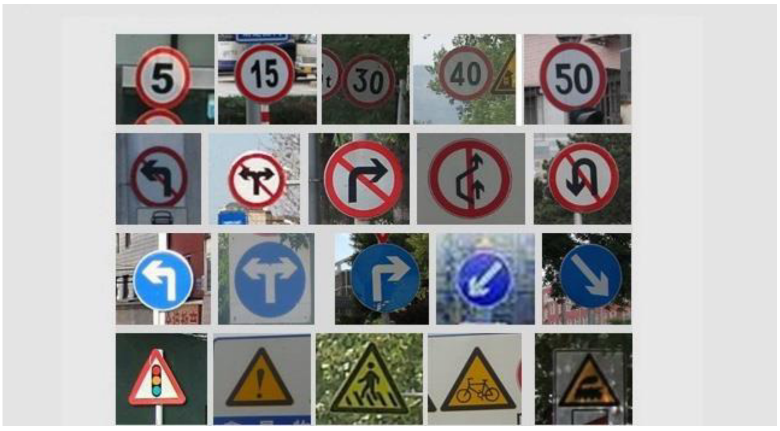
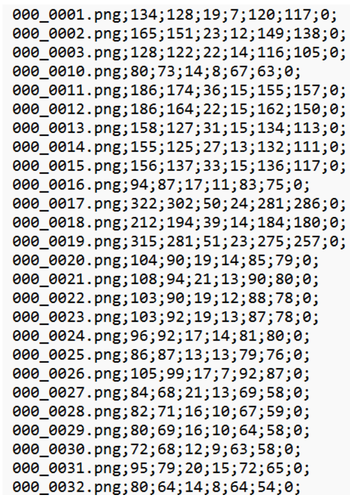


Figure 3 Example of the training set pictures from TSRD

The label file format is not a YOLO model, so I need to write a python script to convert it into a txt label format which is in 4.0 part.



```
000_0001.png;134;128;19;7;120;117;0;
000_0002.png;165;151;23;12;149;138;0;
000_0003.png;128;122;22;14;116;105;0;
000_0010.png;80;73;14;8;67;63;0;
000_0011.png;186;174;36;15;155;157;0;
000_0012.png;186;164;22;15;162;150;0;
000_0013.png;158;127;31;15;134;113;0;
000_0014.png;155;125;27;13;132;111;0;
000_0015.png;156;137;33;15;136;117;0;
000_0016.png;94;87;17;11;83;75;0;
000_0017.png;322;302;50;24;281;286;0;
000_0018.png;212;194;39;14;184;180;0;
000_0019.png;315;281;51;23;275;257;0;
000_0020.png;104;90;19;14;85;79;0;
000_0021.png;108;94;21;13;90;80;0;
000_0022.png;103;90;19;12;88;78;0;
000_0023.png;103;92;19;13;87;78;0;
000_0024.png;96;92;17;14;81;80;0;
000_0025.png;86;87;13;13;79;76;0;
000_0026.png;105;99;17;7;92;87;0;
000_0027.png;84;68;21;13;69;58;0;
000_0028.png;82;71;16;10;67;59;0;
000_0029.png;80;69;16;10;64;58;0;
000_0030.png;72;68;12;9;63;58;0;
000_0031.png;95;79;20;15;72;65;0;
000_0032.png;80;64;14;8;64;54;0;
```

Figure 4 Example of training labels from TSRD

2.3 Convolution Neural Network

The principle of human vision is as follows: start with raw signal intake (pupil intake of pixels Pixels), then do preliminary processing (some cells in the cerebral cortex find edges and directions), and then abstract (the brain determines that the shape of the object in front of you is a circle Shaped), and then further abstracted (the brain further determines that the object is a balloon) [19]. The source of inspiration for deep learning algorithms such as CNN and RNN is to imitate the characteristics of the human brain, construct a multi-layer neural network, identify primary image features at the lower level, and form higher-level features with several lower-level features, and finally pass through multiple levels. Combinations, and finally make a classification at the top level.

CNN is structurally divided into: convolutional layer (feature extraction), pooling layer (reduced to prevent overfitting), fully connected layer (output result).

The principle of the convolutional layer is to use the convolution kernel (filter) to multiply and sum the local area of the input data element by element. The source of the convolution kernel matrix is set based on the features we want to find.

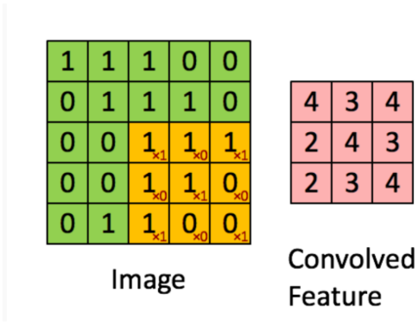


Figure 5: Convolution operation diagram [7]

$$(f * g)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)\, g(x-i, y-j) \qquad (2.3.1)$$

In the formula, f is the input data, g is the filter, and (f*g) represents the result of the convolution operation. The convolution kernel can slide according to the stride (usually 1) during the convolution process, and calculate the sum of the product of the elements at each position. After the convolution operation, the part where the features overlap will get a larger value, and the part that does not overlap will be 0.

The pooling operation is to replace a certain area of the image with a certain value. The purpose is to reduce the amount of parameters, reduce the dimension, remove redundant information, compress features, simplify the network complexity, reduce the amount of calculation, reduce memory consumption, etc.

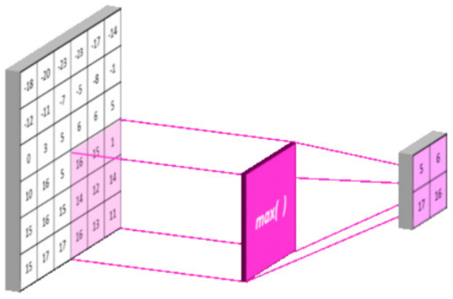


Figure 6: pooling operation

After the pooling operation, the characteristic data of a part of the area can be aggregated together, and they can be represented by calculating their average or maximum value.

Fully connected layers (FC) play the role of "classifier" in the entire convolutional neural network. After convolution and pooling operations, many feature data are obtained, and the fully connected layer will classify, combine, and finally judge.

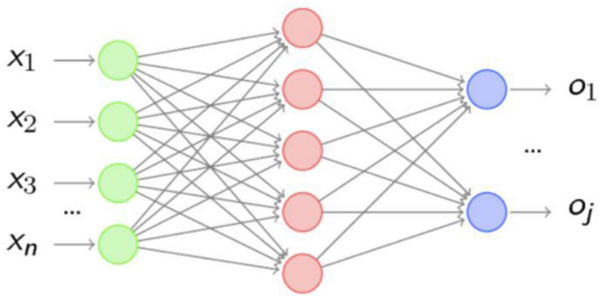


Figure 7: FC example

2.4 YOLOV5

Currently, YOLO has become one of the most outstanding algorithm models in the field of image processing. It is different from Fast R-CNN in that YOLO regards the target detection task as a regression problem by predicting the category and bounding box of the target in a single forward pass, which means that YOLO can cope with scenarios that require fast response such as automatic tasks such as driving. So for the road sign recognition task, YOLO is the most suitable model.

YOLO is a supervised training model that requires human-labeled data and provides images with anchor boxes to support training. After learning through a convolutional neural network architecture, a bounding box is generated to predict the object, which is the head of YOLO.

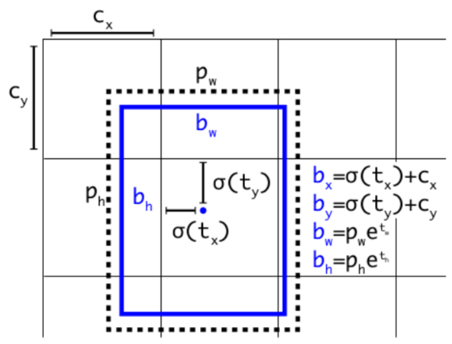
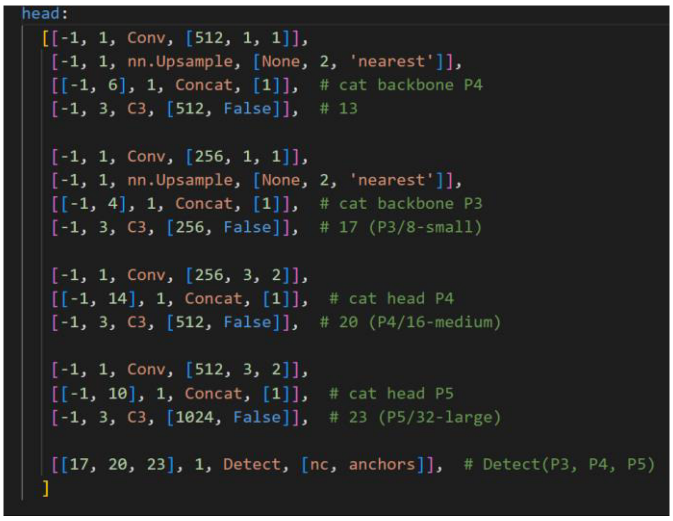


Figure 8: Anchor box and bounding box conversion diagram [10]

As shown, the blue one is the bounding box to be predicted, and the black dotted box is the Anchor box. x and y are the center coordinates of the bounding box, and h and w are the height and width of the bounding box, respectively.

In the code of the head part of YOLOV5S, you can see that the model will process the data through the convolutional layer and the C3 layer before generating the anchor frame. The purpose of this process is to fuse the bottom layer features with the top layer features in order to detect targets at different scales.



```
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 9: The head code of YOLOV5s [18]

YOLOV5 has made some improvements based on YOLOV4, and it performs better in coco object detection.

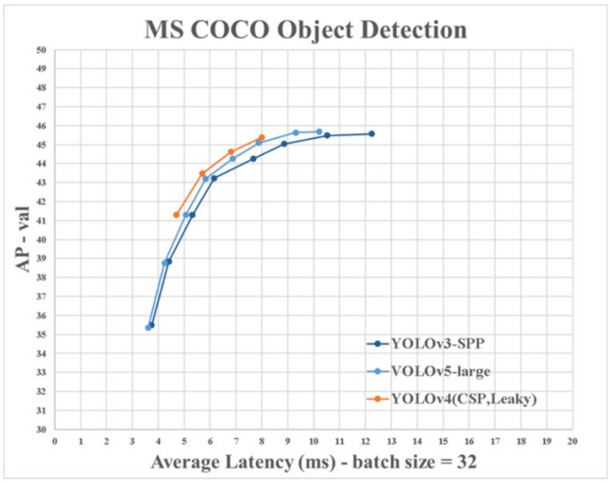


Figure 10: Performance comparison of several versions of yolo [8]

yolov5 is divided into four versions, small (s), medium (m), large (l) and extra large (x). Their mAP and model size are gradually improved. In this project, I am using the yolov5s version.

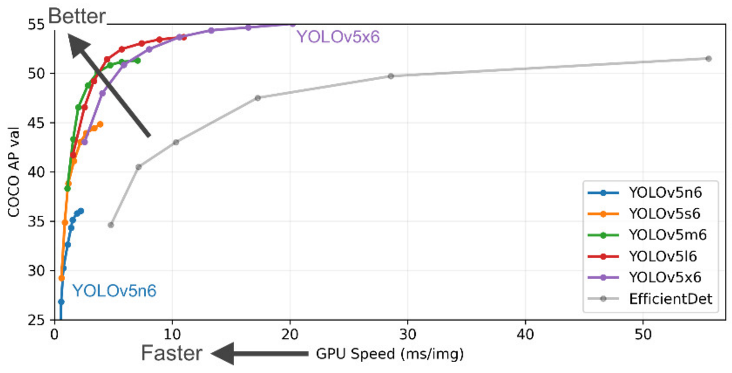


Figure 11: Comparison of several yolo versions[9]

The network structure of yolov5 are as follows:

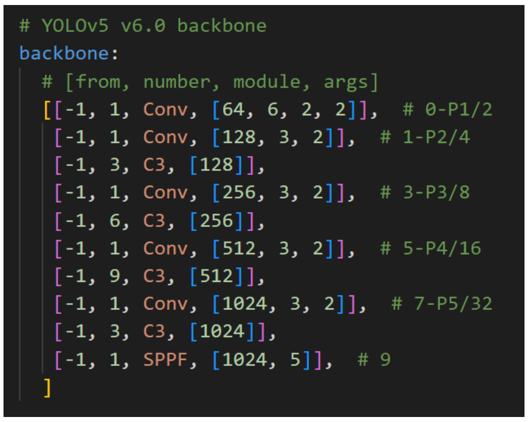
Input: Mosaic data enhancement, adaptive anchor box calculation, adaptive image scaling

Backbone: New CSP-Darknet53

Neck: SPPF +PAN structure

Prediction: GIOU_Loss

The core component is a convolutional neural network that amalgamates and shapes image characteristics at various levels of detail. The core structure in yolov5 remains mostly consistent when compared to v4, with only a minor alteration introduced in yolov5 post-version 6.0. This change involves substituting the initial layer of the network (originally the Focus module) with a 6x6 size convolutional layer. While theoretically equivalent, for specific legacy GPU devices (and their corresponding optimization algorithms), employing a 6x6 convolutional layer proves to be more efficient than the previously utilized Focus module.



```
# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

Figure 12: The backbone code of YOLOV5s

The neck of YOLOV5 consists of a series of layers that mix and combine image features to pass them on for prediction. Compared with the v4 version, v5 replaces SPP with SPPF, which improves efficiency. In YOLO, the SPPF layer is integrated into the backbone part, behind the convolutional layer.

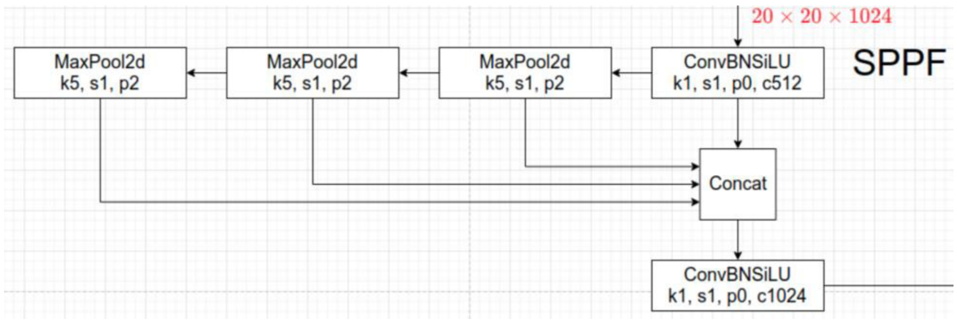


Figure 13: SPPF structure [10]

After network processing, YOLO will send the predicted results to NMS (non maximum suppression), and finally get the final output box result. NMS is to remove some redundant frames, the schematic diagram is as follows:

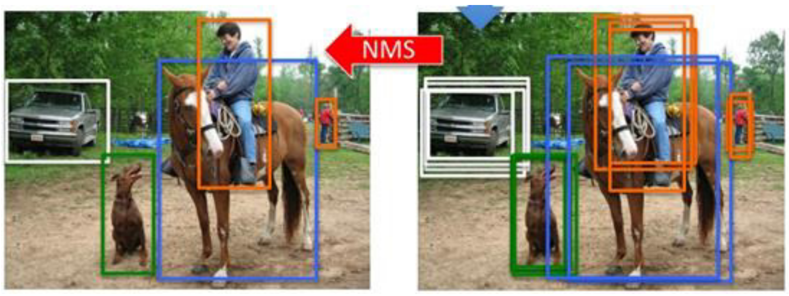


Figure 14: NMS effect display

NMS processing needs to calculate the Intersection over Union (IoU) between all bounding boxes, as shown in the figure below [11].

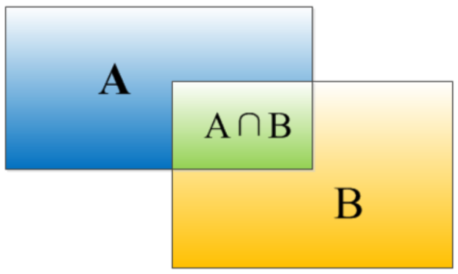


Figure 15: IoU example

$$IoU = \frac{Area(A \cap B)}{Area(A \cup B)} \qquad (2.4.1)$$

When the intersection of AB is greater than a certain threshold, the model will determine that the two are the same bounding box and remove one of them.

Finally, there are several important evaluation indicators: precision and recall.

$$Precision = \frac{TP}{TP+FP} \qquad (2.4.2)$$

$$Recall = \frac{TP}{TP+FN} \qquad (2.4.3)$$

Among them, T and F represent correct and wrong predictions respectively, and P and N represent positive samples and negative samples respectively. Precision and Recall are usually a pair of contradictory performance metrics, because when the recall rate of the model is too high, it means that the model will classify most samples as positive samples, but it will include false negative samples, so the precision will be low. On the contrary, when the accuracy is high, it means that the model has a strong ability to classify positive samples, but its sensitivity to negative samples is poor or even unable to classify negative samples. So we draw the precision and recall rate as a PR curve, which presents the trade-off between the precision and the recall rate of different thresholds, and helps to find the best threshold to maximize these two indicators [12]. The average precision of a class can be abstractly understood as the area of the PR curve, and the Mean Average Precision(mAP) of the training is the average value of AP of all categories, which is also an important indicator for evaluating model performance.

3 Optimization

3.1 attention

When dealing with detection tasks, there are many complex scenarios in the data set, in which the targets we need are often not easy to find, which will affect the convergence process of the training.

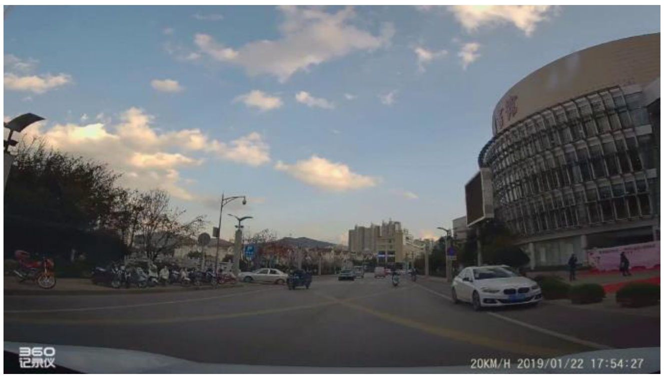


Figure 16: Detection data example

As shown, The traffic signs in the picture account for a very small proportion of the entire scene. In this case, it is very difficult for the convolutional neural network to obtain effective features. In order to optimize this process, I tried adding attention blocks to YOLO. During the process, I tried Squeeze-and-Excitation (SE)[13] and Coordinate Attention (CA)[14]. This part mainly introduces the method ideas, and the specific implementation and comparison results will be carried out in the fourth part.

3.1.1 SEnet

SEnet mainly focuses on the relationship between channels, automatically obtains the importance of each feature channel through learning, and then improves useful features and suppresses features that are not very useful for the current task according to this importance. The principle of SE module is as follows.

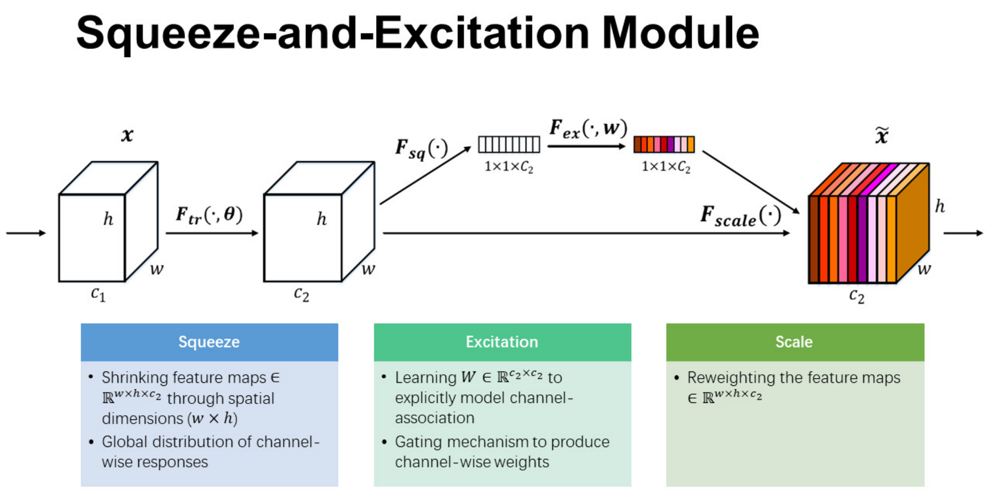

Squeeze-and-Excitation Module
x
$F_{tr}(\cdot,\theta)$
$F_{sq}(\cdot)$
$1\times1\times C_2$
$F_{ex}(\cdot,w)$
$1\times1\times C_2$
$\tilde{x}$
$F_{scale}(\cdot)$
h
w
c_1
c_2
Squeeze
• Shrinking feature maps ∈ $\mathbb{R}^{w\times h\times c_2}$ through spatial dimensions ($w\times h$)
• Global distribution of channel-wise responses
Excitation
• Learning $W\in\mathbb{R}^{c_2\times c_2}$ to explicitly model channel-association
• Gating mechanism to produce channel-wise weights
Scale
• Reweighting the feature maps ∈ $\mathbb{R}^{w\times h\times c_2}$

Figure 17: A Squeeze-and-Excitation module [13]

The data processed by the convolutional layer of YOLO and the CSPDarknet53 layer obtains certain features and is used as input x, and SE performs a Squeeze operation on it, compresses features along the spatial dimension, and encodes the entire spatial feature in a channel into a global feature , this real number has a global receptive field to some extent, and the number of output channels is equal to the number of input feature channels. For example, the feature map with shape (1, 32, 32, 10) is compressed into (1, 1, 1, 10). This operation is usually implemented using global average pooling.

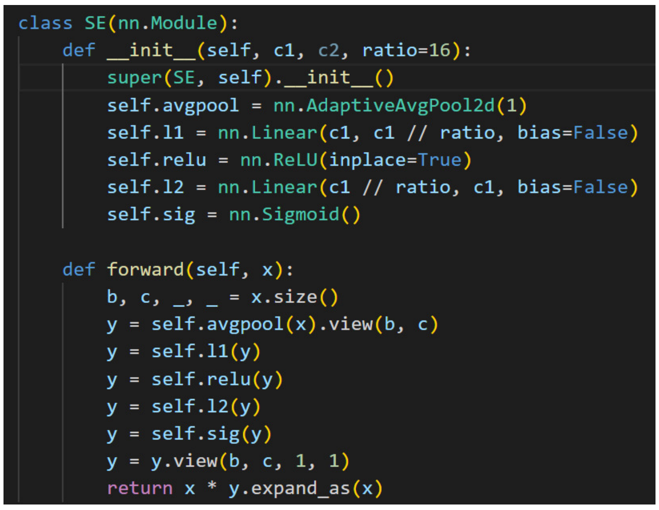
After obtaining the global description features, the Excitation operation is performed to capture the relationship between feature channels:

$$s = F_{ex}(z,W) = \sigma(g(z,W)) = \sigma(W_2 ReLU(W_1 z)) \qquad (3.1.1.1)\ [13]$$

In the formula, $\sigma(f(x))$ is the Sigmoid activation function, W_1 and W_2 are the neural network weight matrix, g is the linear transformation, and $F_{ex}()$ is the output of the external feature extraction function. The end of this operation generates a weight for each feature channel, that is, the learned activation value of each channel (sigmoid activation, the value is between 0 and 1).

The last is the operation of Scale. The weight of the Excitation output is regarded as the importance of each feature channel after feature selection, and then weighted to the previous features by multiplication channel by channel to complete the recalibration of the original features in the channel dimension, so that the model is more discriminative to the characteristics of each channel.

The Pytorch frame based code implementation is as follows.



```
class SE(nn.Module):
    def __init__(self, c1, c2, ratio=16):
        super(SE, self).__init__()
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.l1 = nn.Linear(c1, c1 // ratio, bias=False)
        self.relu = nn.ReLU(inplace=True)
        self.l2 = nn.Linear(c1 // ratio, c1, bias=False)
        self.sig = nn.Sigmoid()

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avgpool(x).view(b, c)
        y = self.l1(y)
        y = self.relu(y)
        y = self.l2(y)
        y = self.sig(y)
        y = y.view(b, c, 1, 1)
        return x * y.expand_as(x)
```

Figure 18: SE block code

Among them, class SE(nn.Module) defines the constructor of the SE module and the forward propagation method.

3.1.2 CA

Channel attention usually ignores location information, which is very important for generating spatially selective attention maps. Different from SE, CA improves the performance of the model by focusing on features at different positions. The main idea is that the feature representations of the model at different spatial locations may have different importance for different tasks. Based on this idea, CA adaptively adjusts the feature weights at each location by introducing coordinate information. So in this project, I also tried to add the CA block to improve the performance of the model.

The CA module encodes channel relationships and long-range dependencies through precise location information. Similar to the SE module, it is also divided into two steps: coordinate information embedding and coordinate attention generation. Its specific structure as shown below.

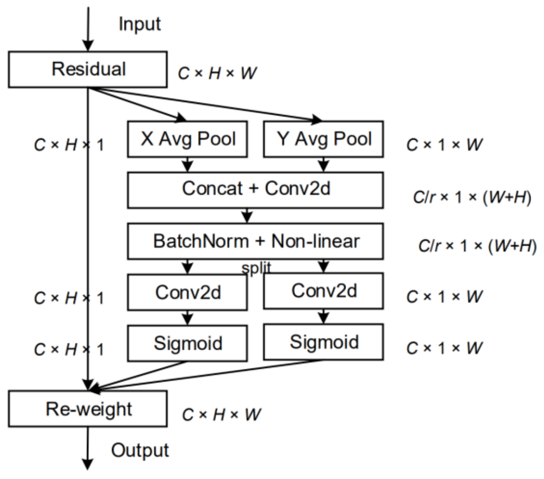


Figure 19: CA module structure [14]

First analyze the idea of coordinate embedding. Global pooling is often used in channel attention to globally encode spatial information as channel descriptors, so it is difficult to preserve positional information. To facilitate the attention module to capture spatial long-range dependencies with precise location information, the authors decompose the global pooling into a one-to-one dimensional feature encoding operation. Specifically, for the input X, first use pooling kernels of size (H,1) and (1,W) to encode each channel along the horizontal and vertical coordinate directions[14]. Therefore, the output of the c-th channel with height h is expressed as follows.

$$z_c^h(h) = \frac{1}{W} \sum_{0 \le i \le W} x_c(h, i) \qquad (3.1.2.1)\ [14]$$

Similarly, the output of the c-th channel of width w is expressed as:

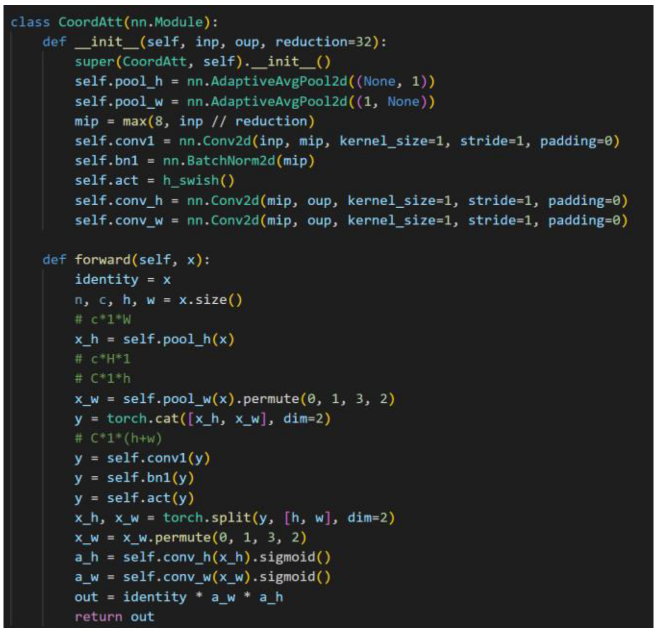
$$z_c^w(h) = \frac{1}{W} \sum_{0 \le i \le W} x_c(h, i) \qquad (3.1.2.2)\ [14]$$

The preceding two operations perform feature aggregation in two spatial dimensions, yielding a duo of orientation-sensitive attention maps. This differs significantly from the manner in which the SE module generates a feature vector. Moreover, these operations enable the attention module to capture extensive spatial relationships in one dimension while conserving exact positional details in the other. This enhancement aids the network in more precisely pinpointing objects of interest. The incorporation of coordinate information corresponds to the X Avg Pool and Y Avg Pool components as depicted in the preceding illustration.

The rest is the coordinate attention generation. Its purpose is to make better use of the information generated in the previous step with a global receptive field and precise location. The final result of this operation is the output of the CA module as follows.

$$y_c(i,j) = x_c(i,j) \times g_c^h(i) \times g_c^w(j) \qquad (3.1.2.3)\ [14]$$

The Pytorch frame based code implementation is as follows.



```python
class CoordAtt(nn.Module):
    def __init__(self, inp, oup, reduction=32):
        super(CoordAtt, self).__init__()
        self.pool_h = nn.AdaptiveAvgPool2d((None, 1))
        self.pool_w = nn.AdaptiveAvgPool2d((1, None))
        mip = max(8, inp // reduction)
        self.conv1 = nn.Conv2d(inp, mip, kernel_size=1, stride=1, padding=0)
        self.bn1 = nn.BatchNorm2d(mip)
        self.act = h_swish()
        self.conv_h = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)
        self.conv_w = nn.Conv2d(mip, oup, kernel_size=1, stride=1, padding=0)

    def forward(self, x):
        identity = x
        n, c, h, w = x.size()
        # c*1*W
        x_h = self.pool_h(x)
        # c*H*1
        # C*1*h
        x_w = self.pool_w(x).permute(0, 1, 3, 2)
        y = torch.cat([x_h, x_w], dim=2)
        # C*1*(h+w)
        y = self.conv1(y)
        y = self.bn1(y)
        y = self.act(y)
        x_h, x_w = torch.split(y, [h, w], dim=2)
        x_w = x_w.permute(0, 1, 3, 2)
        a_h = self.conv_h(x_h).sigmoid()
        a_w = self.conv_w(x_w).sigmoid()
        out = identity * a_w * a_h
        return out
```

Figure 20: CA block code

Among them, class CoordAtt(nn.Module) defines the constructor of the CA module and the forward propagation method.

3.2 Data augmentation

The dataset of the road sign recognition task contains a large number of similar pictures, which may cause the training process to overfit to the training data, and the robustness is weak. At the same time, due to the relatively small amount of data in this data set, for recognition tasks, the training results may lack generalization ability and reduce the actual use ability of the model.

Figure 21: TSRD example

In order to get a better classification under limited data, I used the data enhancement method provided by Python's imgaug[15] to strengthen the training set, so as to alleviate the above problems and improve mAP.

The imgaug library helps users augment images for machine learning projects by transforming a set of input images into a new, larger set of slightly altered images.

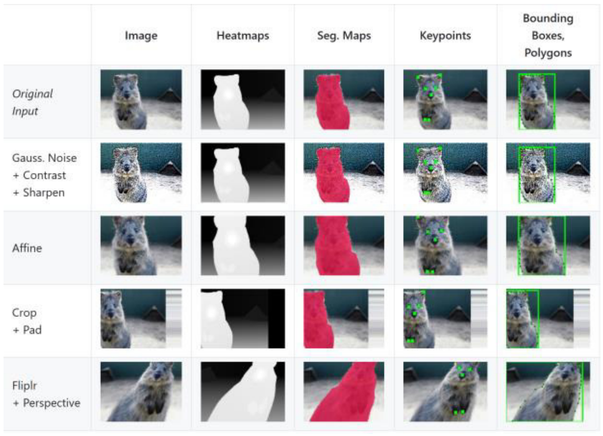
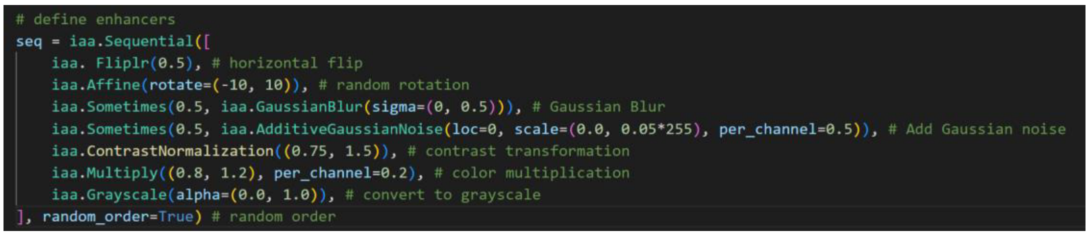


Figure 22: Data Augmentation Comparison Example [16]

In this project, I mainly tried the following methods: horizontal flip, random rotation, Gaussian Blur, Add Gaussian noise, contrast transformation, color multiplication, convert to grayscale, random order.



```
# define enhancers
seq = iaa.Sequential([
    iaa. Fliplr(0.5), # horizontal flip
    iaa.Affine(rotate=(-10, 10)), # random rotation
    iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=(0, 0.5))), # Gaussian Blur
    iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5)), # Add Gaussian noise
    iaa.ContrastNormalization((0.75, 1.5)), # contrast transformation
    iaa.Multiply((0.8, 1.2), per_channel=0.2), # color multiplication
    iaa.Grayscale(alpha=(0.0, 1.0)), # convert to grayscale
], random_order=True) # random order
```

Figure 22: Data Augmentation Code implementation

In the code example, I defined a data enhancer to process all images through loop traversal to obtain an enhanced dataset.

3.3 dhash

In the dataset used, I found that there are a large number of suspected identical pictures, and it is undoubtedly meaningless for the training process to provide the model with the same training data instead of data with subtle differences. To alleviate this problem, I try to use dhash algorithm [17].

Dhash is a hashing algorithm whose purpose is to find two pictures with very small differences. In the image hashing algorithm, the accuracy and speed of dhash are relatively excellent. Its implementation methods are divided into zooming pictures, converting grayscale images, calculating average grayscale, and calculating hash strings. The specific algorithm process is listed in the table below.

Table 1: dhash algorithm process

Step	Content
Resize	Uniformly scale the picture size to 16*16, and get a total of 256 pixels.
Convert to grayscale	Convert non-single-channel images to single-channel grayscale images for the purpose of unifying standards.
Calculating average grayscale	Calculate the average gray level of the obtained grayscale image.
Calculating hash strings	Start from the first pixel to compare it with the average grayscale. If it is larger, add “1” to the hash string, and if it is smaller, add “0”. Each image gets a separate hash string.
Compare	Compare the hash strings of each picture, if the difference value is less than a certain threshold (usually 10), it is judged as the same picture and deleted.

4 Project implementation

4.1 Project environment

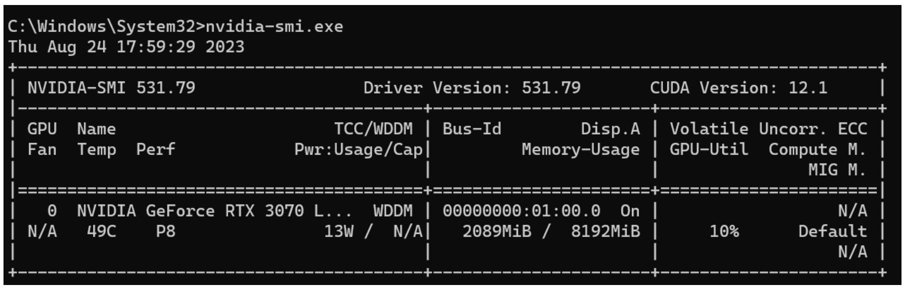
Local device information reference:

Table 2: Device information

Device	Version
Operating system	Windows 11
CPU	12th Gen Intel(R) Core(TM) i7-12700H 2.70GHz
GPU	Nvidia RTX3070 Laptop
RAM	16.0GB

Table 3: Conda virtual environment version

Conda	23.1.0
Python	3.9.16
Pytorch	2.0.1+cu117
Opencv	4.8.0



```
C:\Windows\System32>nvidia-smi.exe
Thu Aug 24 17:59:29 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 531.79                 Driver Version: 531.79       CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                      TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3070 L...  WDDM | 00000000:01:00.0  On |                  N/A |
| N/A   49C    P8               13W /  N/A|   2089MiB /  8192MiB |     10%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

Figure 23: CUDA information

4.2 Dataset processing

The training label in YOLO format requires the label of each picture to be stored in a separate txt file. However, all the data tags given by the TSRD dataset are stored in a txt file. In this case, I need to write a python script to convert it to YOLO format.

First check the content of the data set, where the first data is the file name, the second and third data are the pixels contained in the length and width of the image, the third and fourth data are the xy coordinates of the upper left corner of the Anchor box, and the first the fifth and sixth data are the xy coordinates of the lower right corner of the Anchor box, and the seventh data is the category number.

The algorithm first calculates the normalization factors dw and dh of the horizontal and vertical coordinates, aiming at normalizing all data. Then make the difference between the upper left and lower right coordinates to obtain the length and width of the anchor box.

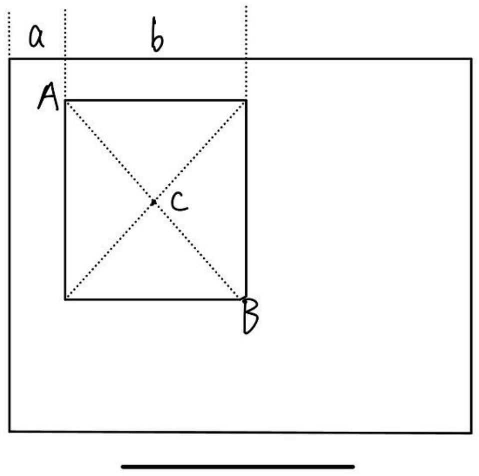


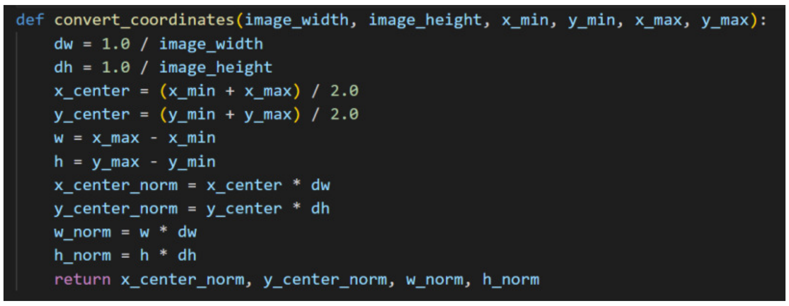
Figure 24: Anchor box example

Suppose the positions of the anchor frame and the picture are as shown in the figure, and C represents the center point of the anchor frame, and a and b represent the length of the line segment respectively, then the horizontal coordinates of C are:

$$C_x = a + \frac{b}{2} = \frac{2a+b}{2} = \frac{a+(a+b)}{2} = \frac{A_X+B_X}{2} \qquad (4.2.1)$$

The same is true for the vertical axis. According to this algorithm, the coordinates of the center point C of the anchor frame can be obtained.

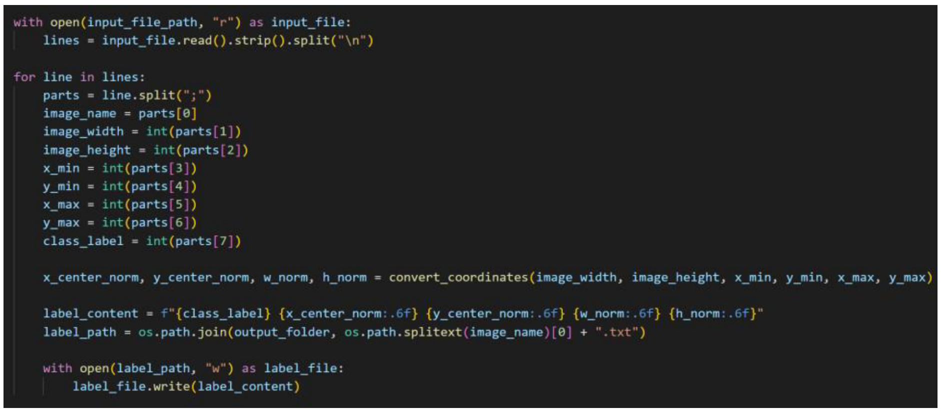
After the analysis is complete, I wrote a python script:



```python
def convert_coordinates(image_width, image_height, x_min, y_min, x_max, y_max):
    dw = 1.0 / image_width
    dh = 1.0 / image_height
    x_center = (x_min + x_max) / 2.0
    y_center = (y_min + y_max) / 2.0
    w = x_max - x_min
    h = y_max - y_min
    x_center_norm = x_center * dw
    y_center_norm = y_center * dh
    w_norm = w * dw
    h_norm = h * dh
    return x_center_norm, y_center_norm, w_norm, h_norm
```

Figure 25: Anchor box coordinate algorithm

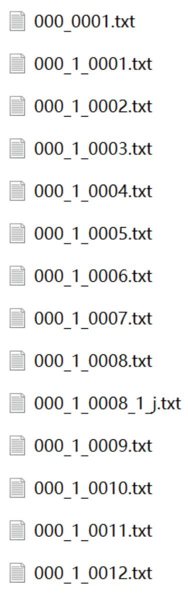
Finally, by traversing the entire folder, the required training label dataset is obtained:



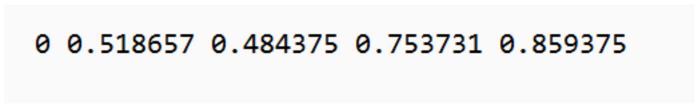
```python
with open(input_file_path, "r") as input_file:
    lines = input_file.read().strip().split("\n")

for line in lines:
    parts = line.split(";")
    image_name = parts[0]
    image_width = int(parts[1])
    image_height = int(parts[2])
    x_min = int(parts[3])
    y_min = int(parts[4])
    x_max = int(parts[5])
    y_max = int(parts[6])
    class_label = int(parts[7])

    x_center_norm, y_center_norm, w_norm, h_norm = convert_coordinates(image_width, image_height, x_min, y_min, x_max, y_max)

    label_content = f"{class_label} {x_center_norm:.6f} {y_center_norm:.6f} {w_norm:.6f} {h_norm:.6f}"
    label_path = os.path.join(output_folder, os.path.splitext(image_name)[0] + ".txt")

    with open(label_path, "w") as label_file:
        label_file.write(label_content)
```

Figure 26

The result is as follows:



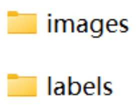
```
000_0001.txt
000_1_0001.txt
000_1_0002.txt
000_1_0003.txt
000_1_0004.txt
000_1_0005.txt
000_1_0006.txt
000_1_0007.txt
000_1_0008.txt
000_1_0008_1_j.txt
000_1_0009.txt
000_1_0010.txt
000_1_0011.txt
000_1_0012.txt
```



```
0 0.518657 0.484375 0.753731 0.859375
```

Figure 27: process result

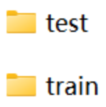
After getting the label data, store the image dataset and label dataset in the root directory of the data:



```
images
labels
```

Figure 28: images and labels list

The training set and test set of image and label data are stored separately:



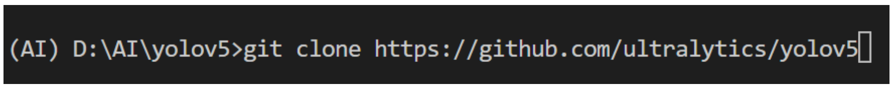
```
test
train
```

Figure 29: internal file structure

After all this is done, the preparation of the data is complete.

4.3 Model cloning and environment configuration

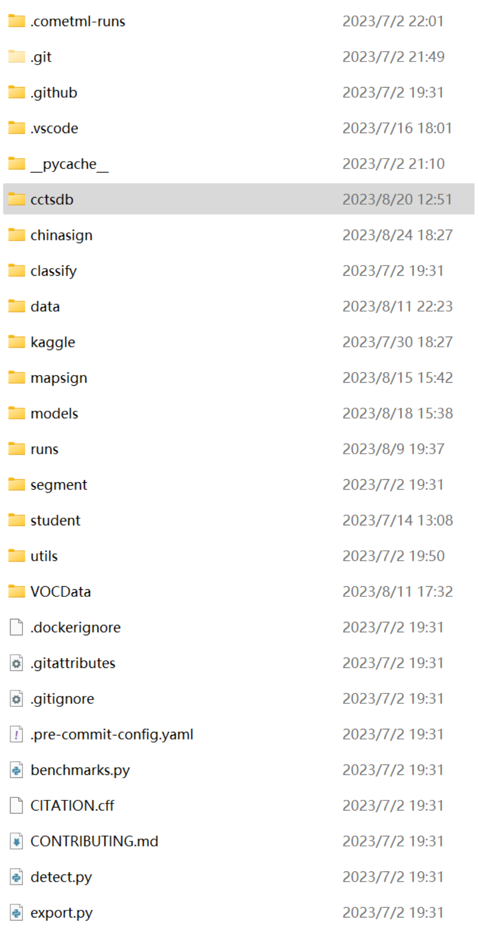
First, we need to clone the official model of the YOLO model using Git.



```
(AI) D:\AI\yolov5>git clone https://github.com/ultralytics/yolov5
```

Figure 30: clone command [18]

We can see that the model is successfully deployed locally.



```
.cometml-runs              2023/7/2 22:01
.git                       2023/7/2 21:49
.github                    2023/7/2 19:31
.vscode                    2023/7/16 18:01
__pycache__                2023/7/2 21:10
cctsdb                     2023/8/20 12:51
chinasign                  2023/8/24 18:27
classify                   2023/7/2 19:31
data                       2023/8/11 22:23
kaggle                     2023/7/30 18:27
mapsign                    2023/8/15 15:42
models                     2023/8/18 15:38
runs                       2023/8/9 19:37
segment                    2023/7/2 19:31
student                    2023/7/14 13:08
utils                      2023/7/2 19:50
VOCData                    2023/8/11 17:32
.dockerignore              2023/7/2 19:31
.gitattributes             2023/7/2 19:31
.gitignore                 2023/7/2 19:31
.pre-commit-config.yaml    2023/7/2 19:31
benchmarks.py              2023/7/2 19:31
CITATION.cff               2023/7/2 19:31
CONTRIBUTING.md            2023/7/2 19:31
detect.py                  2023/7/2 19:31
export.py                  2023/7/2 19:31
```

Figure 31: model file

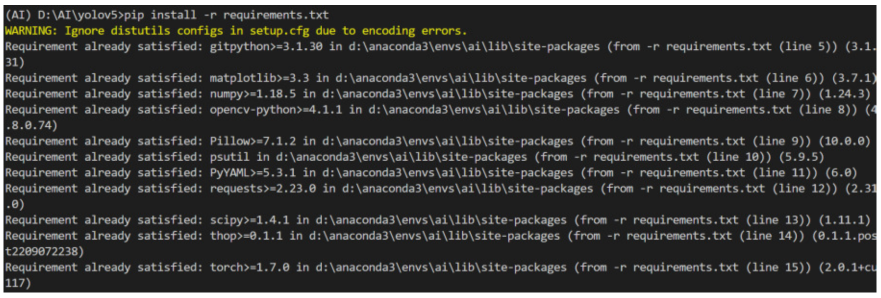
Some of the main model folders and files are described below.

Table 4: Introduction to yolo model file

Data	Store the training yaml file, can also

	store the training data set
Models	The model weight file is stored, network block files as well as the export.py and yolo.py files, which can export the model weight to different formats such as onnx, etc.
Utils	Store third-party function files and plug-ins
Yolov5s.pt	The initial weight file provided by the author of yolo, the network effect trained as the initial weight will be better.
Train.py	Files used to start training
Detect.py	Files used for startup detection

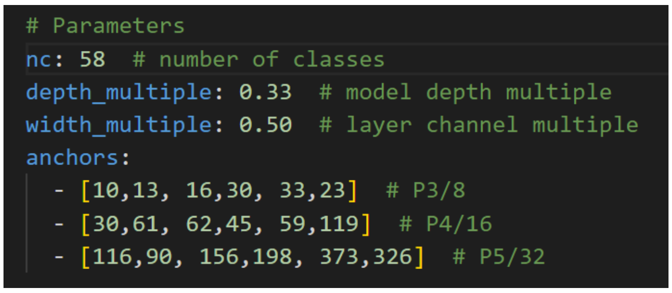
Then install the model dependencies:



```
(AI) D:\AI\yolov5>pip install -r requirements.txt
WARNING: Ignore distutils configs in setup.cfg due to encoding errors.
Requirement already satisfied: gitpython>=3.1.30 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 5)) (3.1.
31)
Requirement already satisfied: matplotlib>=3.3 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 6)) (3.7.1)
Requirement already satisfied: numpy>=1.18.5 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 7)) (1.24.3)
Requirement already satisfied: opencv-python>=4.1.1 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 8)) (4
.8.0.74)
Requirement already satisfied: Pillow>=7.1.2 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 9)) (10.0.0)
Requirement already satisfied: psutil in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 10)) (5.9.5)
Requirement already satisfied: PyYAML>=5.3.1 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 11)) (6.0)
Requirement already satisfied: requests>=2.23.0 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 12)) (2.31
.0)
Requirement already satisfied: scipy>=1.4.1 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 13)) (1.11.1)
Requirement already satisfied: thop>=0.1.1 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 14)) (0.1.1.pos
t2209072238)
Requirement already satisfied: torch>=1.7.0 in d:\anaconda3\envs\ai\lib\site-packages (from -r requirements.txt (line 15)) (2.0.1+cu
117)
```

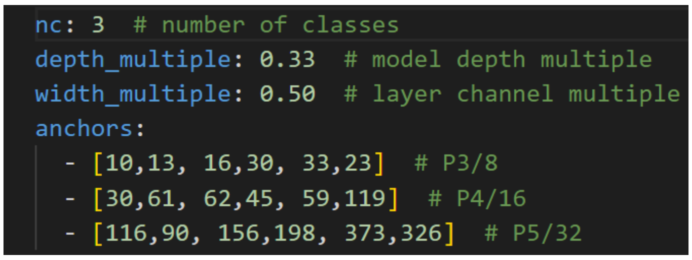
Figure 32: Installing

Then modify the nc(number of classes) value in yolov5s.yaml.



```
# Parameters
nc: 58  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

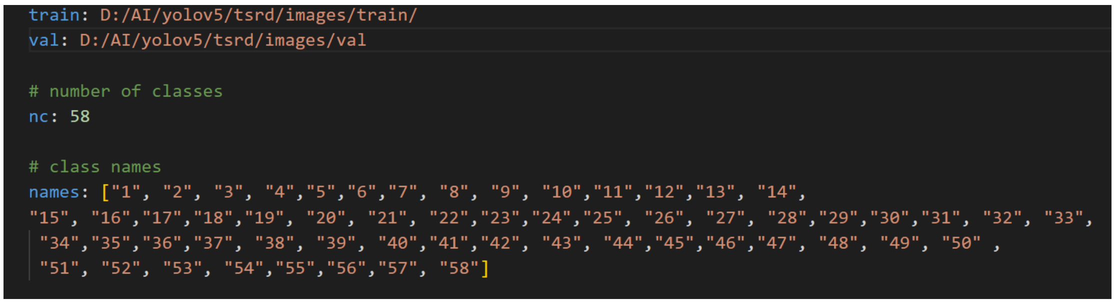
Figure 33: for TSRD



```
nc: 3  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

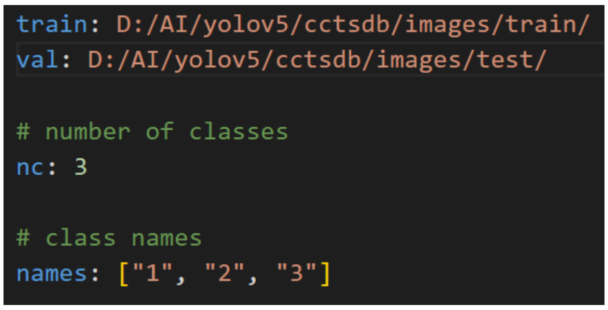
Figure 34: for CCTSDB

Create training yaml files and save to the data folder.



```
train: D:/AI/yolov5/tsrd/images/train/
val: D:/AI/yolov5/tsrd/images/val

# number of classes
nc: 58

# class names
names: ["1", "2", "3", "4","5","6","7", "8", "9", "10","11","12","13", "14",
"15", "16","17","18","19", "20", "21", "22","23","24","25", "26", "27", "28","29","30","31", "32", "33",
 "34","35","36","37", "38", "39", "40","41","42", "43", "44","45","46","47", "48", "49", "50" ,
 "51", "52", "53", "54","55","56","57", "58"]
```

Figure 35: TSRD.yaml



```
train: D:/AI/yolov5/cctsdb/images/train/
val: D:/AI/yolov5/cctsdb/images/test/

# number of classes
nc: 3

# class names
names: ["1", "2", "3"]
```

Figure 36: CCTSDB.yaml

So far, the environment configuration and model configuration are all completed.

4.4 Training

4.4.1 Baseline of CCTSDB

Since the training volume of this project is not particularly large, in order to improve the training speed, I used the weight file of yolov5s as the baseline. Under the same conditions, only the attention layer was changed for comparison.

Set training information:

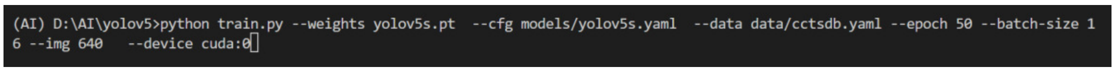
epoch = 50

batch size = 16

device using GPU

weight using yolov5s

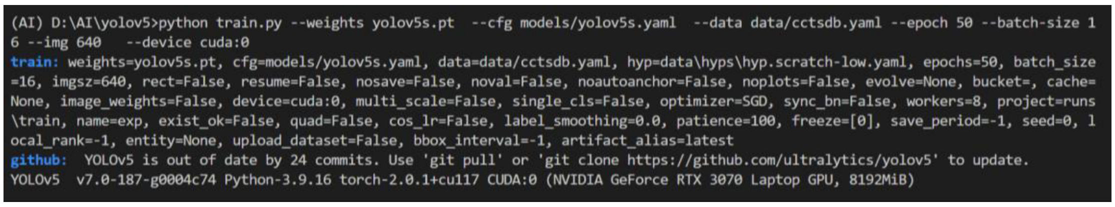
Enter the command in the terminal.



```
(AI) D:\AI\yolov5>python train.py --weights yolov5s.pt  --cfg models/yolov5s.yaml  --data data/cctsdb.yaml --epoch 50 --batch-size 1
6 --img 640   --device cuda:0
```

Figure 37: training command line

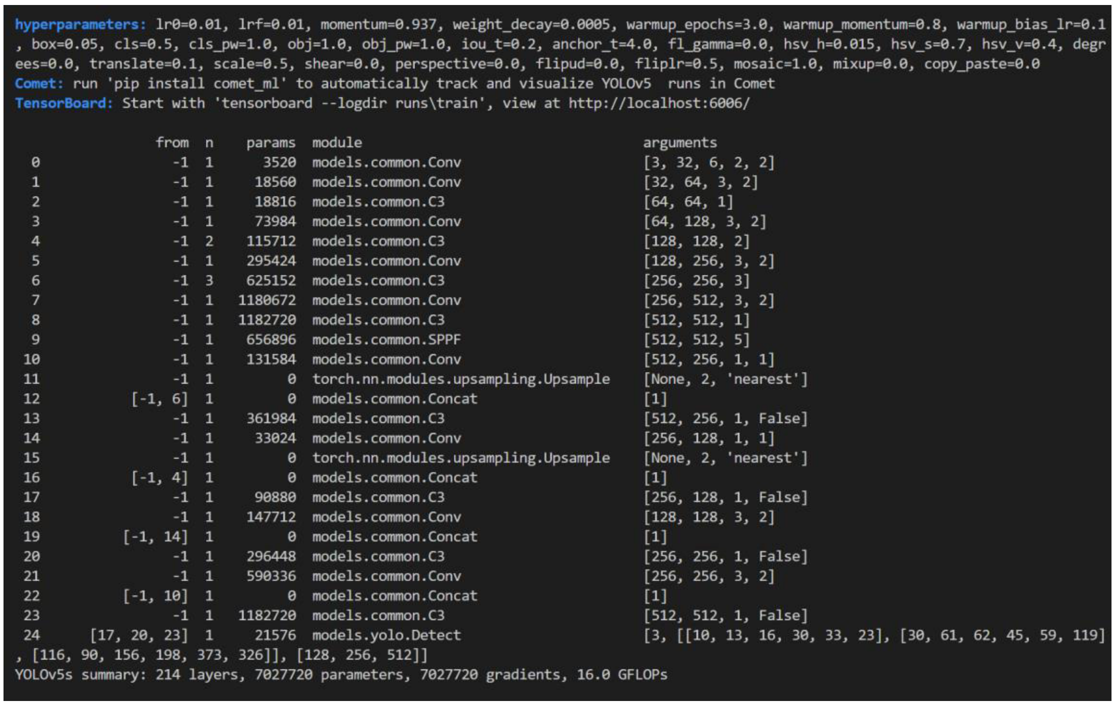
The model starts to load parameters:



```
(AI) D:\AI\yolov5>python train.py --weights yolov5s.pt  --cfg models/yolov5s.yaml  --data data/cctsdb.yaml --epoch 50 --batch-size 1
6 --img 640   --device cuda:0
train: weights=yolov5s.pt, cfg=models/yolov5s.yaml, data=data/cctsdb.yaml, hyp=data\hyps\hyp.scratch-low.yaml, epochs=50, batch_size
=16, imgsz=640, rect=False, resume=False, nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None, bucket=, cache=
None, image_weights=False, device=cuda:0, multi_scale=False, single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=runs
\train, name=exp, exist_ok=False, quad=False, cos_lr=False, label_smoothing=0.0, patience=100, freeze=[0], save_period=-1, seed=0, l
ocal_rank=-1, entity=None, upload_dataset=False, bbox_interval=-1, artifact_alias=latest
github:  YOLOv5 is out of date by 24 commits. Use 'git pull' or 'git clone https://github.com/ultralytics/yolov5' to update.
YOLOv5  v7.0-187-g0004c74 Python-3.9.16 torch-2.0.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3070 Laptop GPU, 8192MiB)
```

Figure 38: model loading

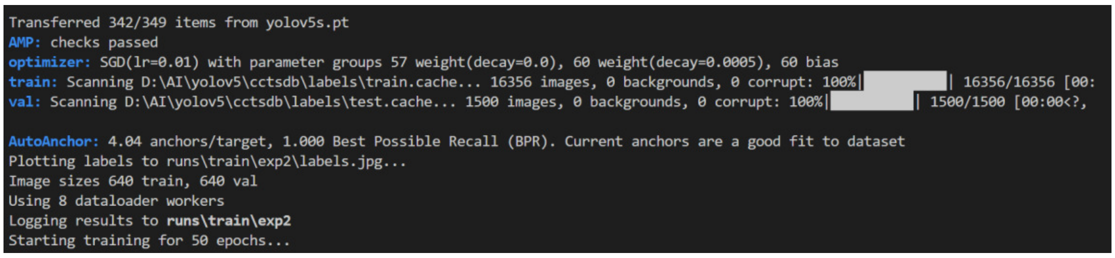
Start loading the training configuration, hyperparameters, comet, tensorboard. Then load the model backbone structure: three-layer loop structure of convolution—C3—SPPF:



```
hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1
, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degr
ees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5  runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs\train', view at http://localhost:6006/

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24      [17, 20, 23]  1     21576  models.yolo.Detect                      [3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119]
, [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
YOLOv5s summary: 214 layers, 7027720 parameters, 7027720 gradients, 16.0 GFLOPs
```

Figure 39: model loading

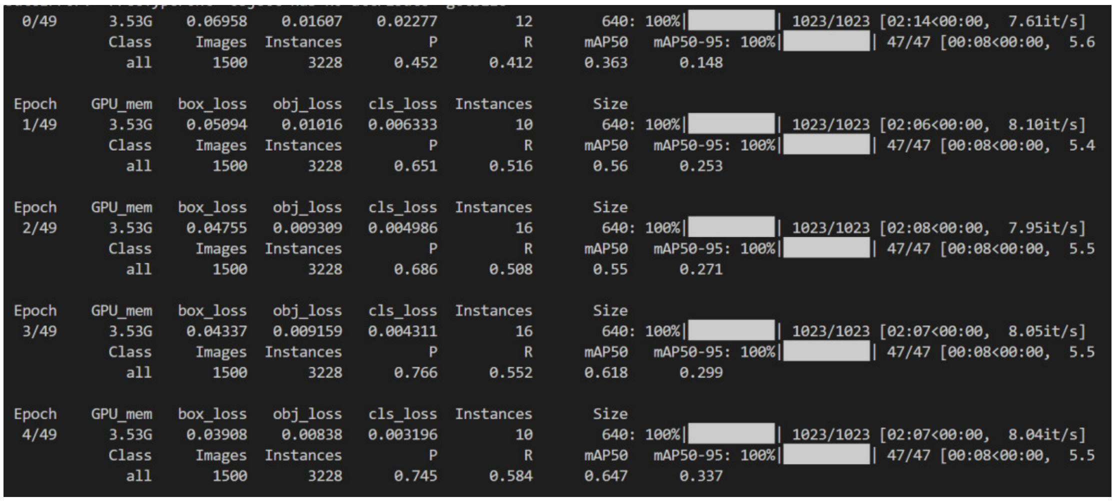
Finally, the model loads AMP, optimizer (SGD), training set and test set. After the verification is passed, the training starts.



```
Transferred 342/349 items from yolov5s.pt
AMP: checks passed
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
train: Scanning D:\AI\yolov5\cctsdb\labels\train.cache... 16356 images, 0 backgrounds, 0 corrupt: 100%|██████████| 16356/16356 [00:
val: Scanning D:\AI\yolov5\cctsdb\labels\test.cache... 1500 images, 0 backgrounds, 0 corrupt: 100%|██████████| 1500/1500 [00:00<?,

AutoAnchor: 4.04 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset
Plotting labels to runs\train\exp2\labels.jpg...
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs\train\exp2
Starting training for 50 epochs...
```

Figure 40: model loading

After starting the training, you can see that each epoch will calculate three loss function values during the training process. At this stage, the model will automatically randomly sample 1023 data from the training set for learning. After getting the weights, use 1500 validation sets to verify the results. During this process, the model will automatically calculate the P value (precision value), r value (recall rate), mAP50 (average mAP with IoU threshold larger than 0.5), mAP50-95 (average mAP when IoU threshold is 0.5 to 0.95, with a step size of 0.05).



```
   0/49      3.53G    0.06958    0.01607    0.02277         12        640: 100%|██████████| 1023/1023 [02:14<00:00,  7.61it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 47/47 [00:08<00:00,  5.6
               all       1500       3228      0.452      0.412      0.363      0.148

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      1/49      3.53G    0.05094    0.01016   0.006333         10        640: 100%|██████████| 1023/1023 [02:06<00:00,  8.10it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 47/47 [00:08<00:00,  5.4
               all       1500       3228      0.651      0.516       0.56      0.253

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      2/49      3.53G    0.04755   0.009309   0.004986         16        640: 100%|██████████| 1023/1023 [02:08<00:00,  7.95it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 47/47 [00:08<00:00,  5.5
               all       1500       3228      0.686      0.508       0.55      0.271

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      3/49      3.53G    0.04337   0.009159   0.004311         16        640: 100%|██████████| 1023/1023 [02:07<00:00,  8.05it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 47/47 [00:08<00:00,  5.5
               all       1500       3228      0.766      0.552      0.618      0.299

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      4/49      3.53G    0.03908    0.00838   0.003196         10        640: 100%|██████████| 1023/1023 [02:07<00:00,  8.04it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 47/47 [00:08<00:00,  5.5
               all       1500       3228      0.745      0.584      0.647      0.337
```

Figure 41: Training start

This process will last about 3 hours. After the training is complete, the results will be automatically saved to the runs\exp folder.

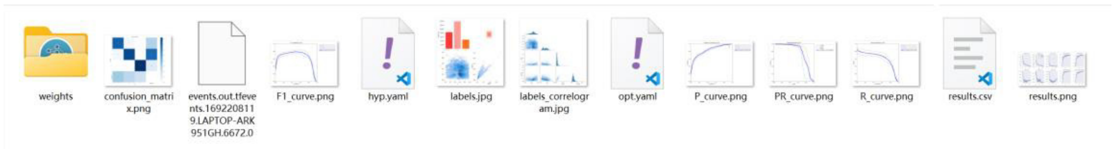
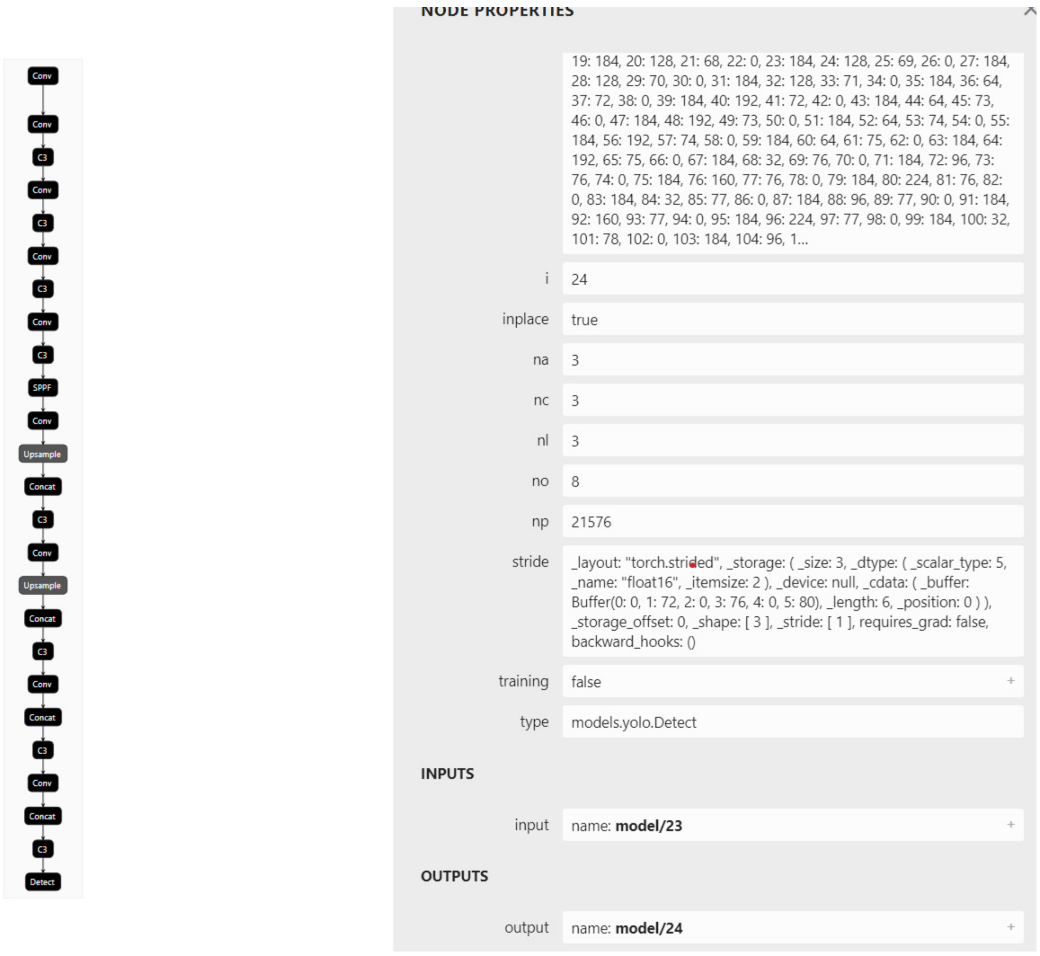


Figure 42: runs folder content

Among them, the weight file best.pt with best training effect and the weight file last.pt of the last epoch are saved in the weight folder.



NODE PROPERTIES

```
          19: 184, 20: 128, 21: 68, 22: 0, 23: 184, 24: 128, 25: 69, 26: 0, 27: 184,
          28: 128, 29: 70, 30: 0, 31: 184, 32: 128, 33: 71, 34: 0, 35: 184, 36: 64,
          37: 72, 38: 0, 39: 184, 40: 192, 41: 72, 42: 0, 43: 184, 44: 64, 45: 73,
          46: 0, 47: 184, 48: 192, 49: 73, 50: 0, 51: 184, 52: 64, 53: 74, 54: 0, 55:
          184, 56: 192, 57: 74, 58: 0, 59: 184, 60: 64, 61: 75, 62: 0, 63: 184, 64:
          192, 65: 75, 66: 0, 67: 184, 68: 32, 69: 76, 70: 0, 71: 184, 72: 96, 73:
          76, 74: 0, 75: 184, 76: 160, 77: 76, 78: 0, 79: 184, 80: 224, 81: 76, 82:
          0, 83: 184, 84: 32, 85: 77, 86: 0, 87: 184, 88: 96, 89: 77, 90: 0, 91: 184,
          92: 160, 93: 77, 94: 0, 95: 184, 96: 224, 97: 77, 98: 0, 99: 184, 100: 32,
          101: 78, 102: 0, 103: 184, 104: 96, 1...
       i  24
 inplace  true
      na  3
      nc  3
      nl  3
      no  8
      np  21576
  stride  _layout: "torch.strided", _storage: ( _size: 3, _dtype: ( _scalar_type: 5,
          _name: "float16", _itemsize: 2 ), _device: null, _cdata: ( _buffer:
          Buffer(0: 0, 1: 72, 2: 0, 3: 76, 4: 0, 5: 80), _length: 6, _position: 0 ) ),
          _storage_offset: 0, _shape: [ 3 ], _stride: [ 1 ], requires_grad: false,
          backward_hooks: ()
training  false
    type  models.yolo.Detect
```

INPUTS

```
   input  name: model/23
```

OUTPUTS

```
  output  name: model/24
```

Figure 43: best.pt file internal structure

The second plot is the confusion matrix. Its diagonal represents the proportion of correctly detected categories for that row. background is a separate class. The background line (FN) represents the original background, which is divided into non-background by pred, and false checks for objects that do not exist. The background column (FP) represents that it was not the background, and was divided into the background by pred, and non-background objects were missed. A confusion matrix is a summary of the prediction results for a classification problem. Aggregating the number of correct and incorrect predictions using count values, broken down by each class, is the crux of the confusion matrix. The confusion matrix shows which parts of the classification model are confused when making predictions. It not only allows us to understand the mistakes made by classification models, but more importantly, which types of errors are occurring. It is this disaggregation of the results that overcomes the limitations of using only classification accuracy.

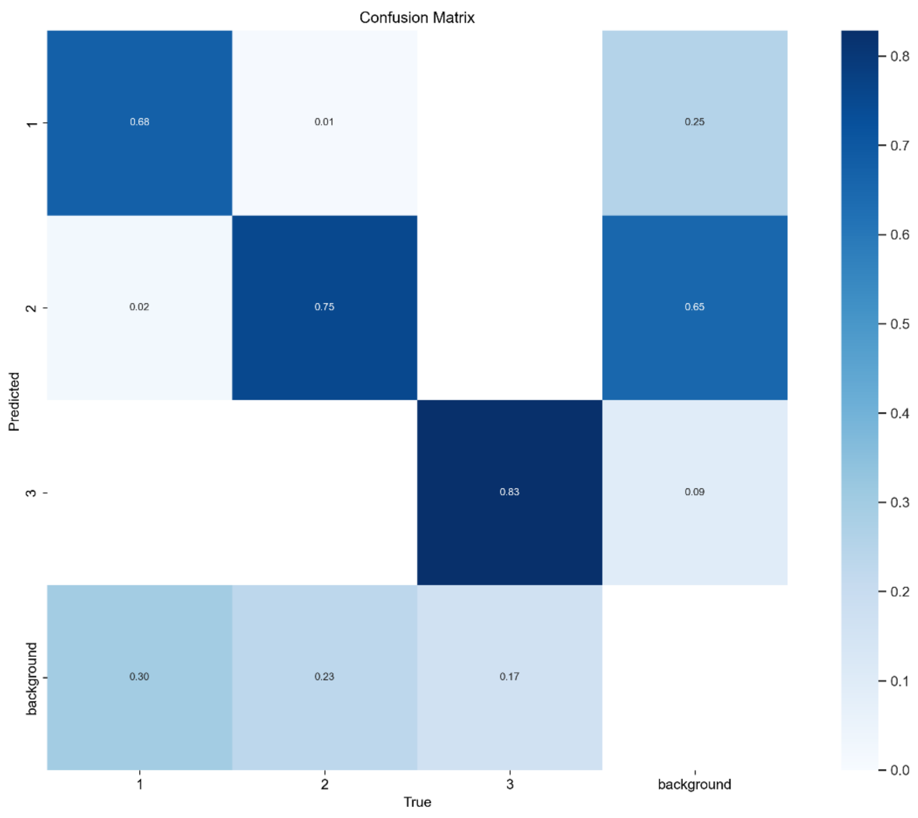


Figure 44: Confusion Matrix Diagram

From the confusion matrix of the training, it can be seen that the last line of background-FN falsely detected a small number of road signs. The class 2 in background FP relatively high, which means many second-type road signs are missed. In the non-background category, it can be seen that the value of the diagonal line is relatively large, and it can be seen that the model can clearly distinguish three categories.

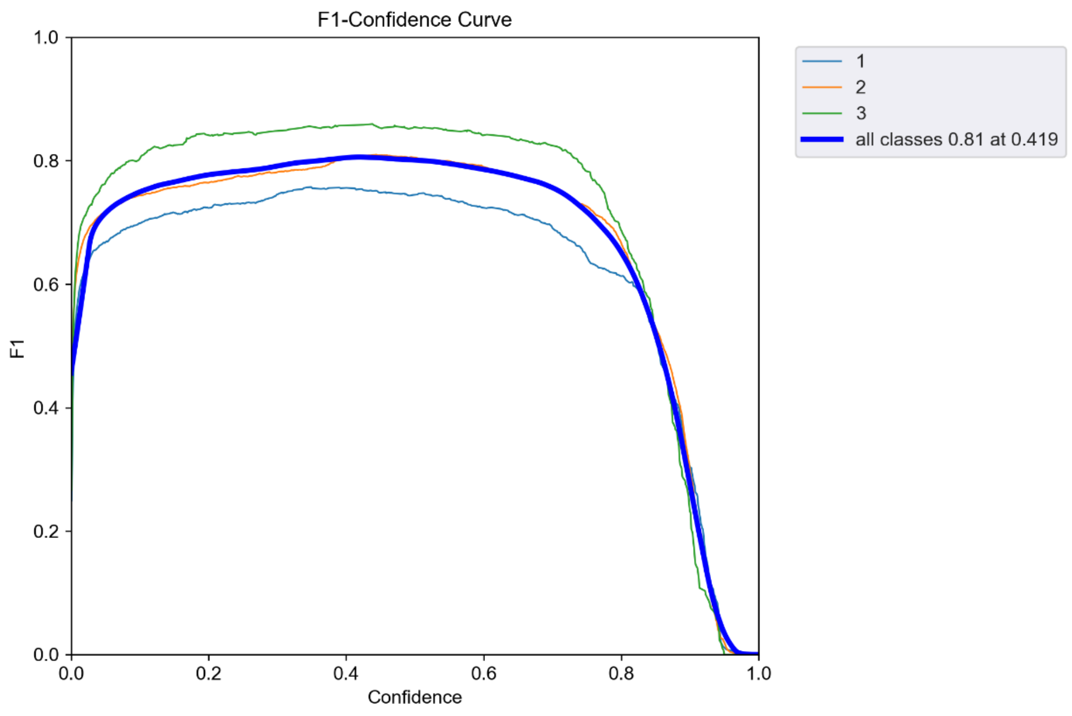


Figure 45: F1 curve

The abscissa of the F1 curve is the confidence threshold:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \qquad (4.4.1.1)$$

From F1 curve, we can see which category can achieve the best F1 score under what threshold. For example, the green line is around 0.2, and the sea blue is around 0.4. The relatively stable peak indicates that the training effect is good.

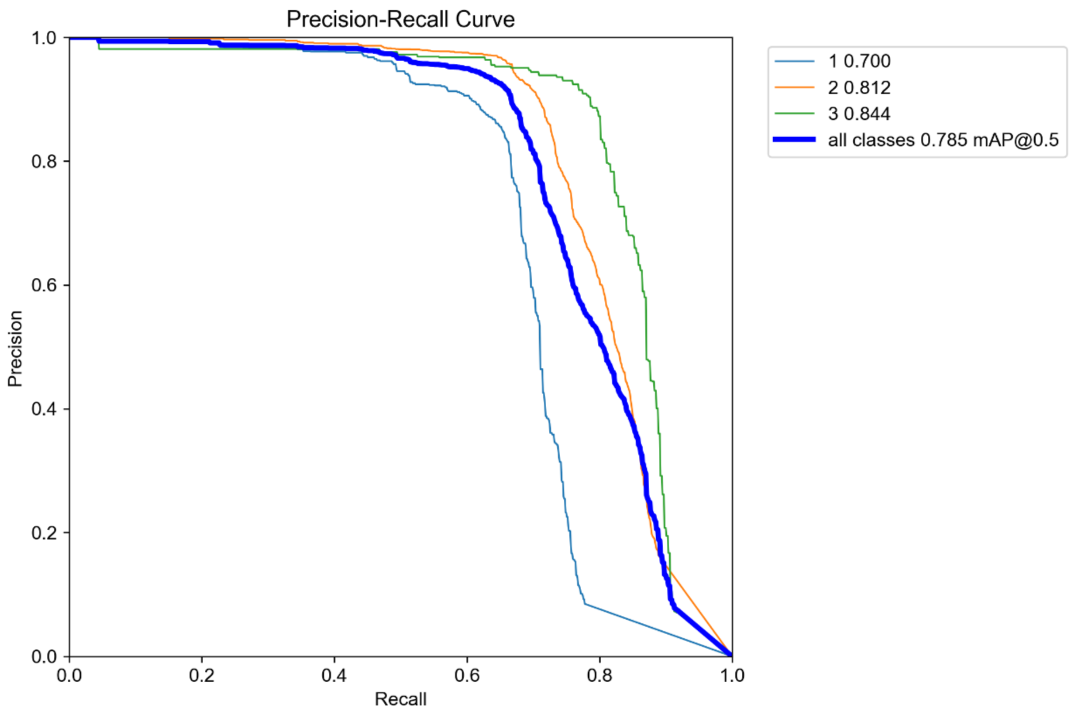


Figure 46: PR curve

The PR curve has been introduced in Section 2.4. Here we can see that the model's detection ability for category 1 is relatively weak, and its detection ability for category 3 is the strongest. This performance can also be seen in the confusion matrix.

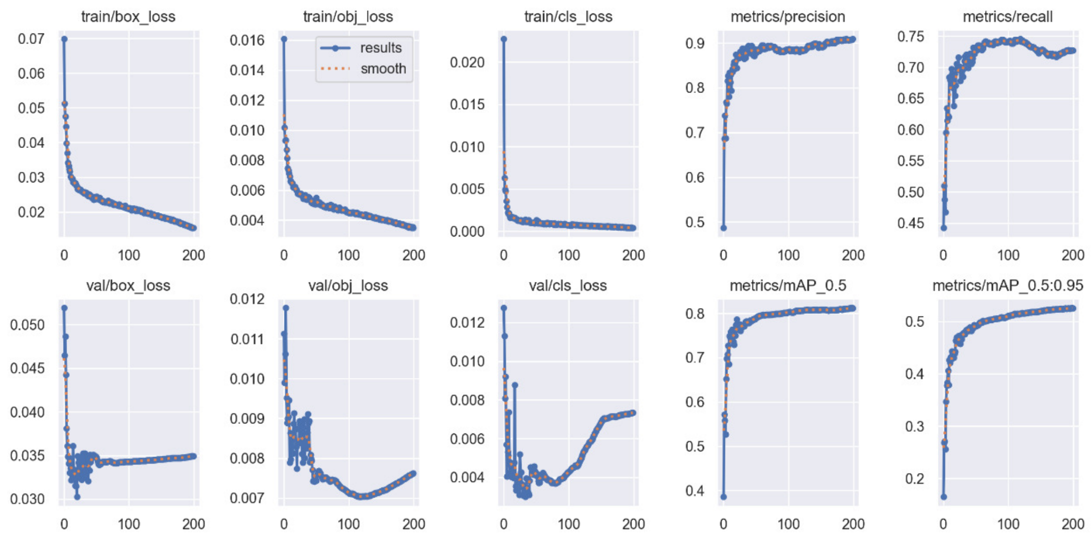
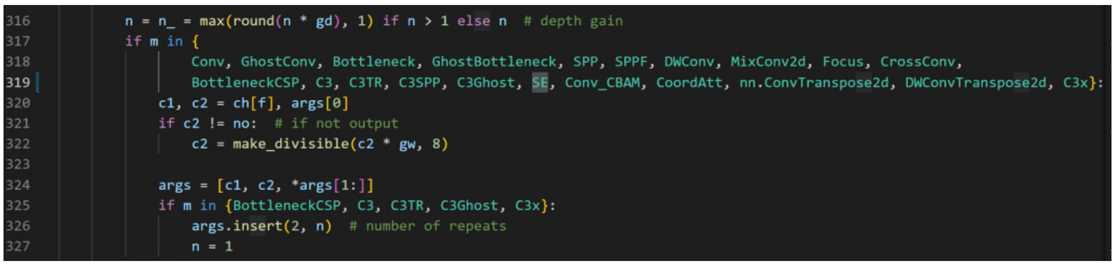


Figure 47: result

It can be seen from the result graph that the training process of the model is relatively stable, and the loss value does not change very disorderly.

4.4.2 Add attention block SE

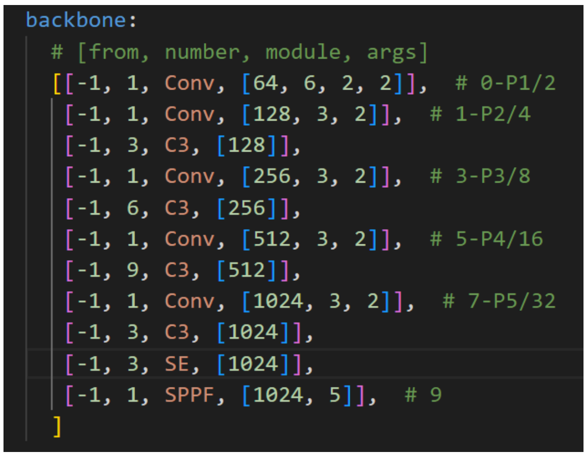
First, we need to add the SE block code in 3.1.1 to the YOLOV5 model, in models/common.py. Next, add the SE module to the parse_model function in models/yolo.py:



```python
        n = n_ = max(round(n * gd), 1) if n > 1 else n  # depth gain
        if m in {
                Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPF, DWConv, MixConv2d, Focus, CrossConv,
                BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, SE, Conv_CBAM, CoordAtt, nn.ConvTranspose2d, DWConvTranspose2d, C3x}:
            c1, c2 = ch[f], args[0]
            if c2 != no:  # if not output
                c2 = make_divisible(c2 * gw, 8)

            args = [c1, c2, *args[1:]]
            if m in {BottleneckCSP, C3, C3TR, C3Ghost, C3x}:
                args.insert(2, n)  # number of repeats
                n = 1
```

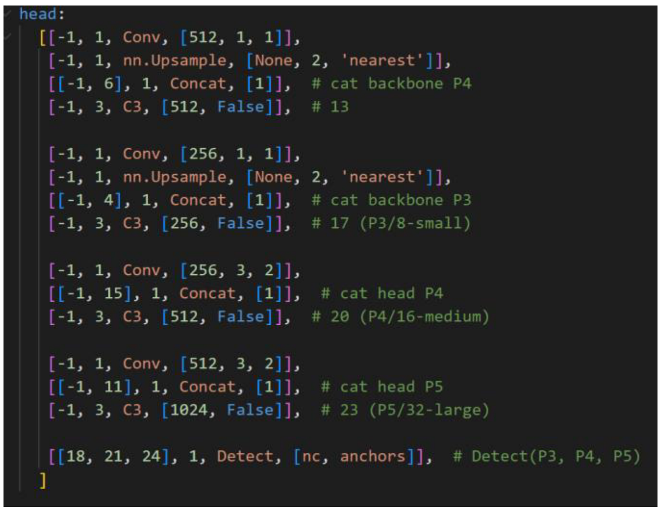
Figure 48: Registration module information

Finally, modify the configuration file yolov5s.yaml, add attention to the last layer of the backbone, and then modify the from coefficients of the two Concats in the head part from [-1, 14], [-1, 10] to [-1 , 15], [-1, 11], the from coefficient of Detect is changed from [17, 20, 23] to [18,21,24]:



```yaml
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 3, SE, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

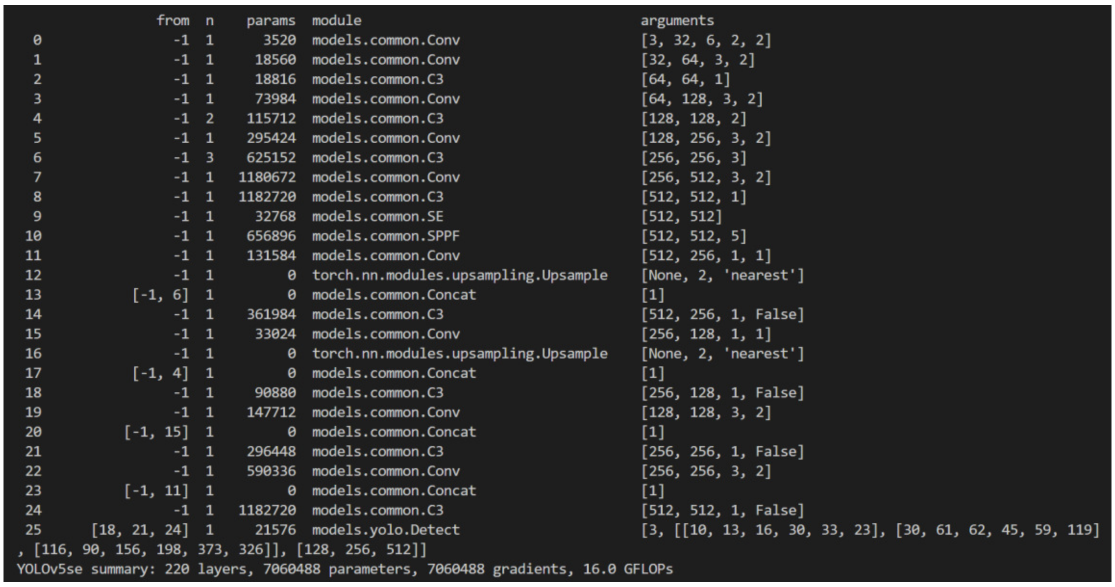
Figure 49: Modify the backbone part



```
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 15], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 11], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[18, 21, 24], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 50: Modify the head part

Start training, we can see that YOLO successfully loaded the SE block. In the figure below, we can see that a layer of SE attention layer is successfully loaded in the ninth line, and the position is between the C3 layer and the SPPF layer. This layer has 32768 parameters with a dimension of 512.



```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1     32768  models.common.SE                        [512, 512]
 10                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 11                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 12                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 13           [-1, 6]  1         0  models.common.Concat                    [1]
 14                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 15                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 16                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 17           [-1, 4]  1         0  models.common.Concat                    [1]
 18                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 19                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 20          [-1, 15]  1         0  models.common.Concat                    [1]
 21                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 22                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 23          [-1, 11]  1         0  models.common.Concat                    [1]
 24                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 25      [18, 21, 24]  1     21576  models.yolo.Detect                      [3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119]
, [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
YOLOv5se summary: 220 layers, 7060488 parameters, 7060488 gradients, 16.0 GFLOPs
```

Figure 51: loading SE block

The training effect is as follows.

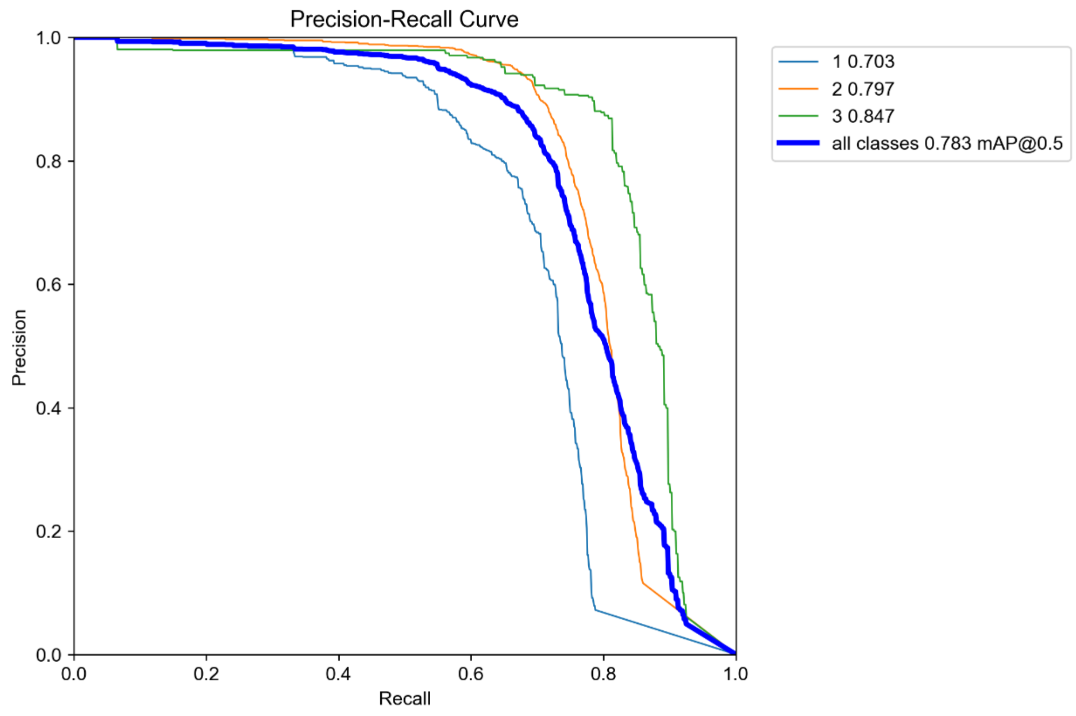


Figure 52: PR curve

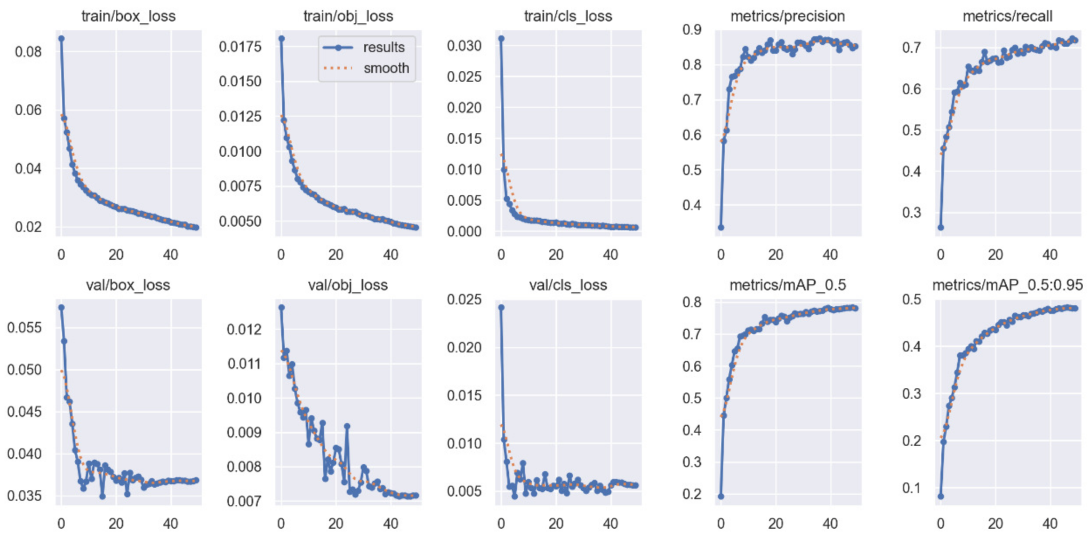
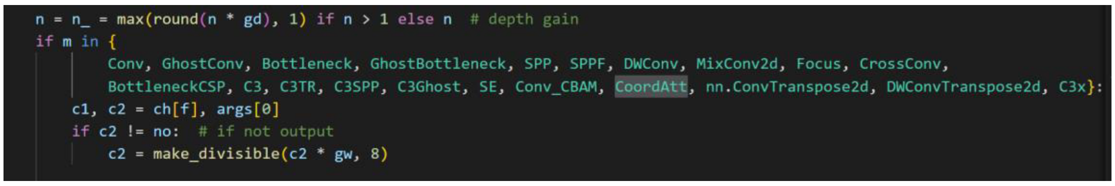


Figure 53: training result

From the PR curve, it can be seen that the SE mechanism did not improve the overall performance of the model, on the contrary, the mAP decreased by 0.002. The result graph shows that the SE module can promote the smooth change of the obj_loss and cls_loss values of the model on the verification set.

4.4.3 Add attention block CA

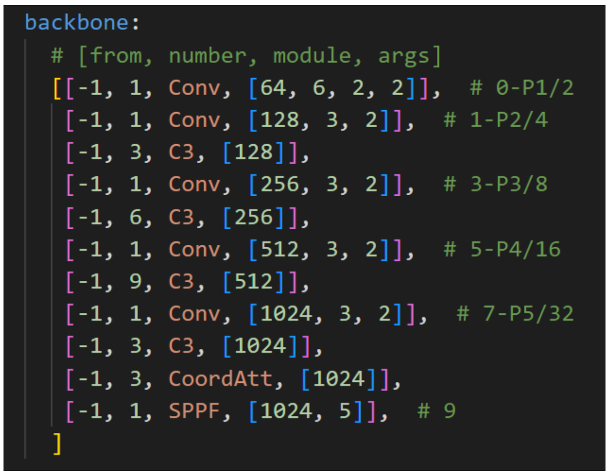
First, we need to add the CA block code in part 3.1.2 to the YOLOV5 model, in models/common.py. Next, add the CA module to the parse_model function in models/yolo.py:



```
n = n_ = max(round(n * gd), 1) if n > 1 else n  # depth gain
if m in {
        Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPF, DWConv, MixConv2d, Focus, CrossConv,
        BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, SE, Conv_CBAM, CoordAtt, nn.ConvTranspose2d, DWConvTranspose2d, C3x}:
    c1, c2 = ch[f], args[0]
    if c2 != no:  # if not output
        c2 = make_divisible(c2 * gw, 8)
```

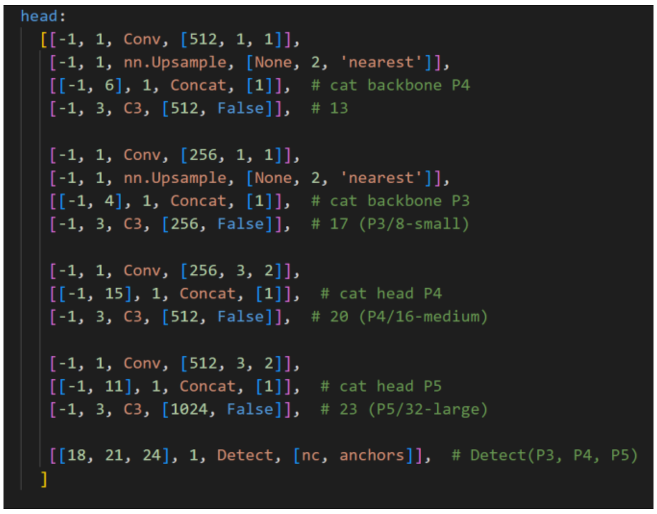
Figure 54: Registration module information

Finally, modify the configuration file yolov5s.yaml, add attention to the last layer of the backbone, and then modify the from coefficients of the two Concats in the head part from [-1, 14], [-1, 10] to [-1 , 15], [-1, 11], the from coefficient of Detect is changed from [17, 20, 23] to [18,21,24]:



```
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 3, CoordAtt, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]
```

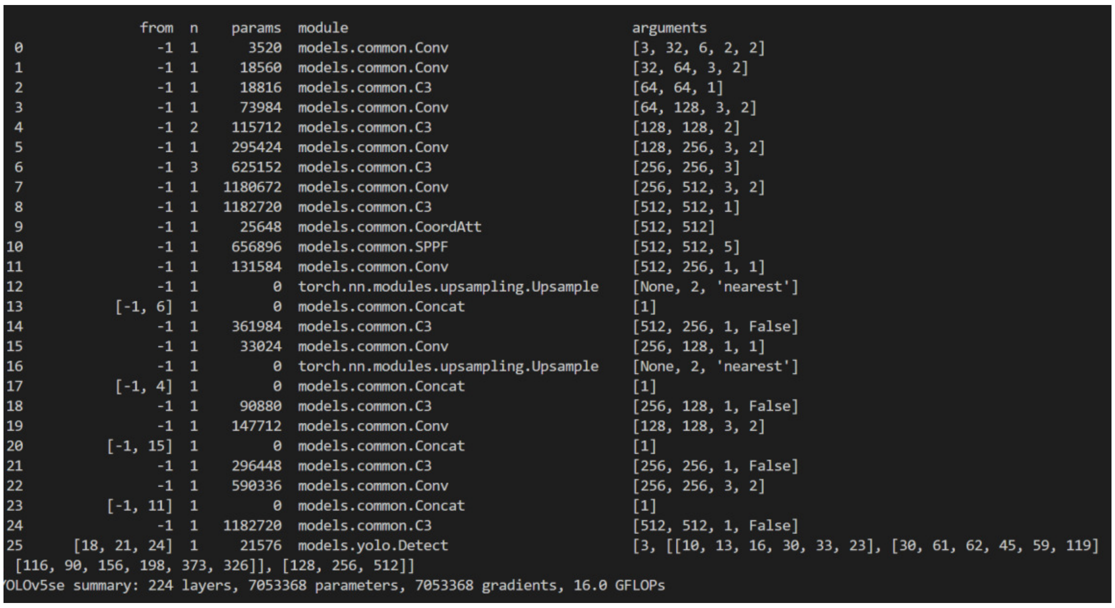
Figure 55: Modify the backbone part



```
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 15], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 11], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[18, 21, 24], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 56: Modify the head part

Start training, we can see that YOLO successfully loaded the CA block. In the figure below, we can see that a layer of CA attention layer is successfully loaded in the ninth line, and the position is between the C3 layer and the SPPF layer. This layer has 25648 parameters with a dimension of 512.



```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1     25648  models.common.CoordAtt                  [512, 512]
 10                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 11                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 12                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 13           [-1, 6]  1         0  models.common.Concat                    [1]
 14                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 15                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 16                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 17           [-1, 4]  1         0  models.common.Concat                    [1]
 18                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 19                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 20          [-1, 15]  1         0  models.common.Concat                    [1]
 21                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 22                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 23          [-1, 11]  1         0  models.common.Concat                    [1]
 24                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 25      [18, 21, 24]  1     21576  models.yolo.Detect                      [3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119]
 [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
OLOv5se summary: 224 layers, 7053368 parameters, 7053368 gradients, 16.0 GFLOPs
```

Figure 57: loading CA block

The training effect is as follows.

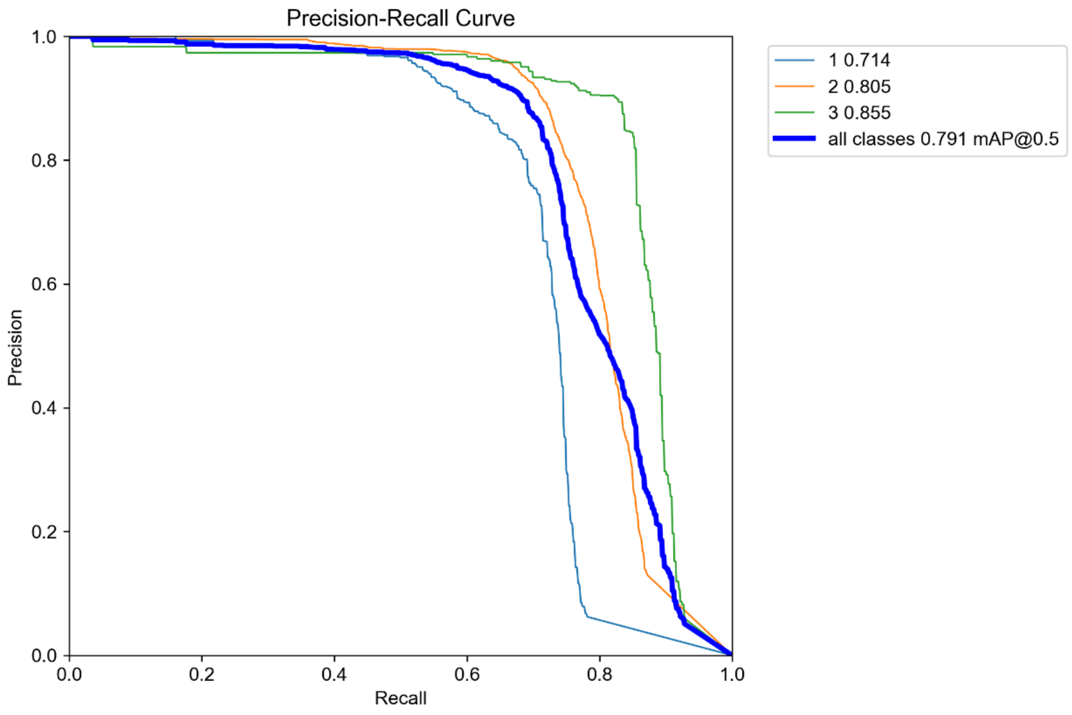


Figure 58: PR curve

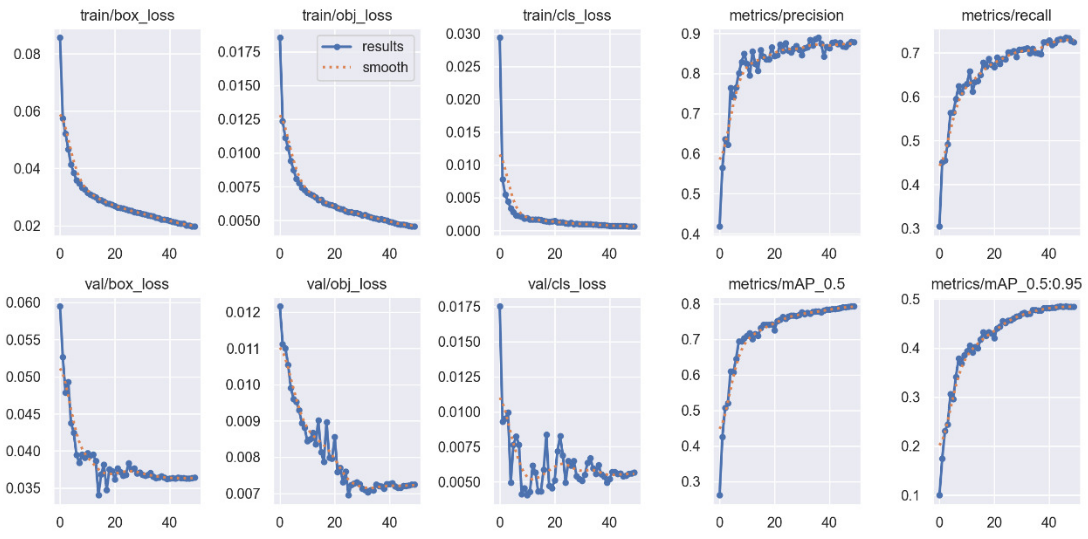


Figure 59: training result

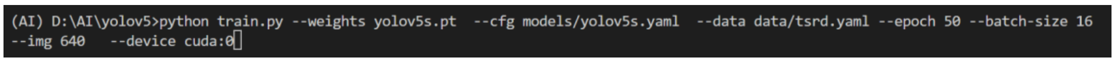
From the PR curve, it can be seen that the addition of the CA module has effectively improved the performance a little, and the mAP of the model has increased by 0.006, but the cls_loss of the model on the verification set fluctuates greatly, and the training process is not smooth.

4.4.4 Baseline of TSRD

The same as the detection task, I still choose YOLOV5s as the baseline for the recognition task, and perform data enhancement on the data set under the same conditions, and observe the results.

Set training information:
epoch = 200
batch size = 16
device using GPU
weight using yolov5s

Enter the command in the terminal.



```
(AI) D:\AI\yolov5>python train.py --weights yolov5s.pt  --cfg models/yolov5s.yaml  --data data/tsrd.yaml --epoch 50 --batch-size 16
--img 640   --device cuda:0
```

Figure 60: training command line

start training. This process is exactly the same as the detection task, so we won’t go into details here, so we directly observe the results:

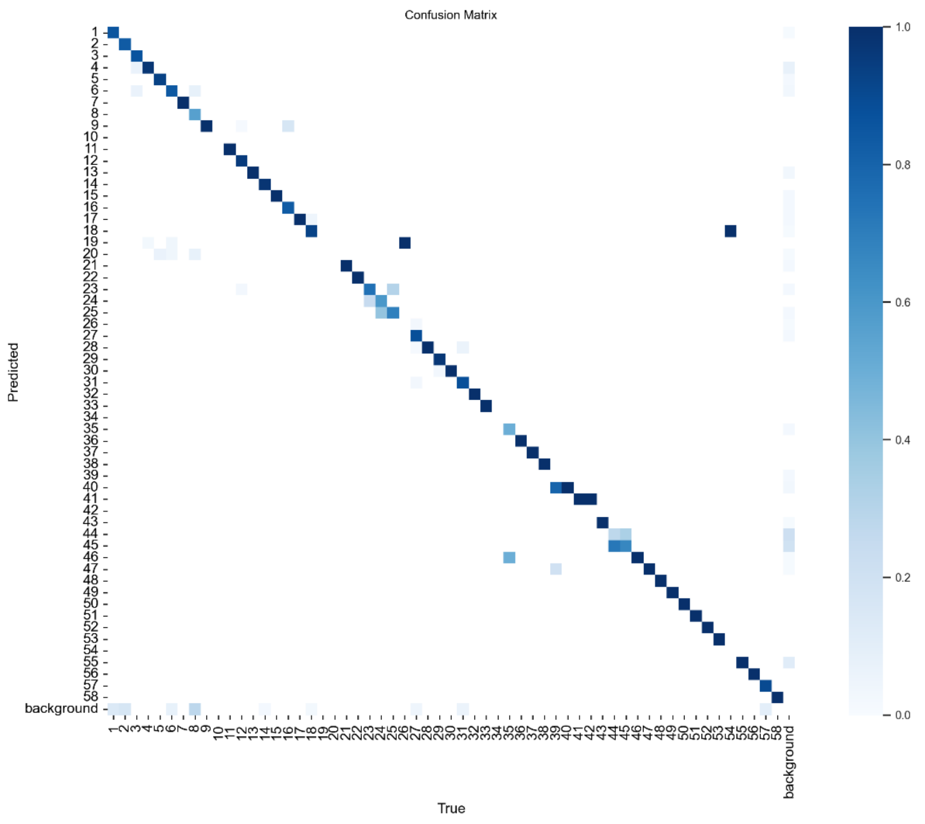


Figure 61: Confusion matrix

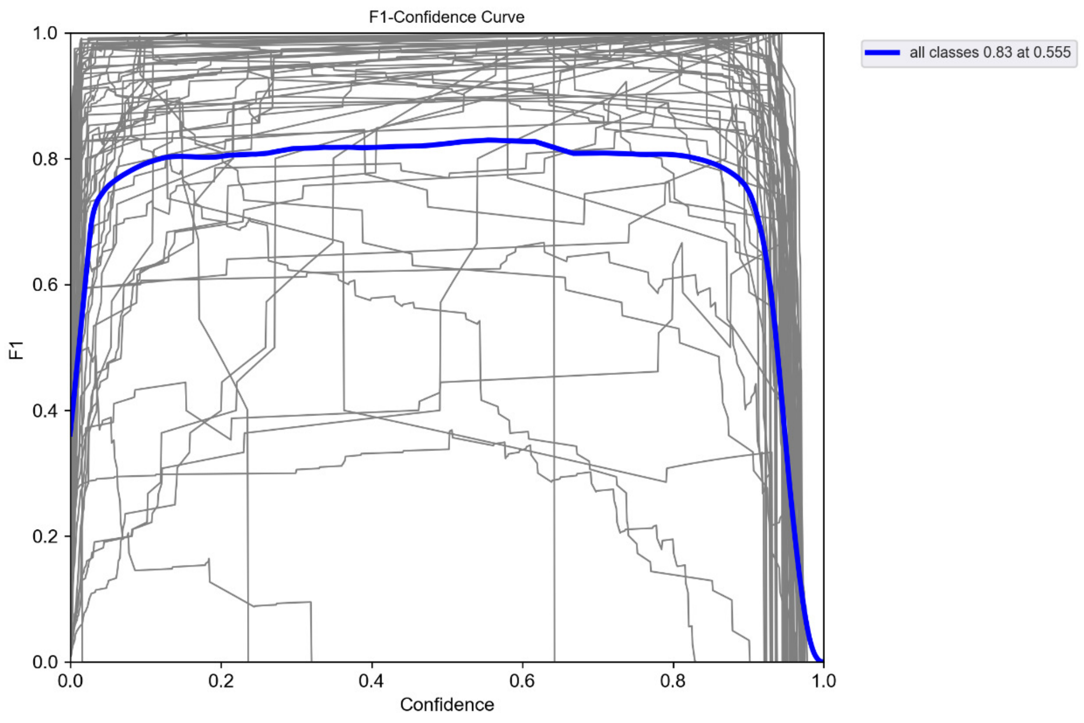


Figure 62: F1 curve

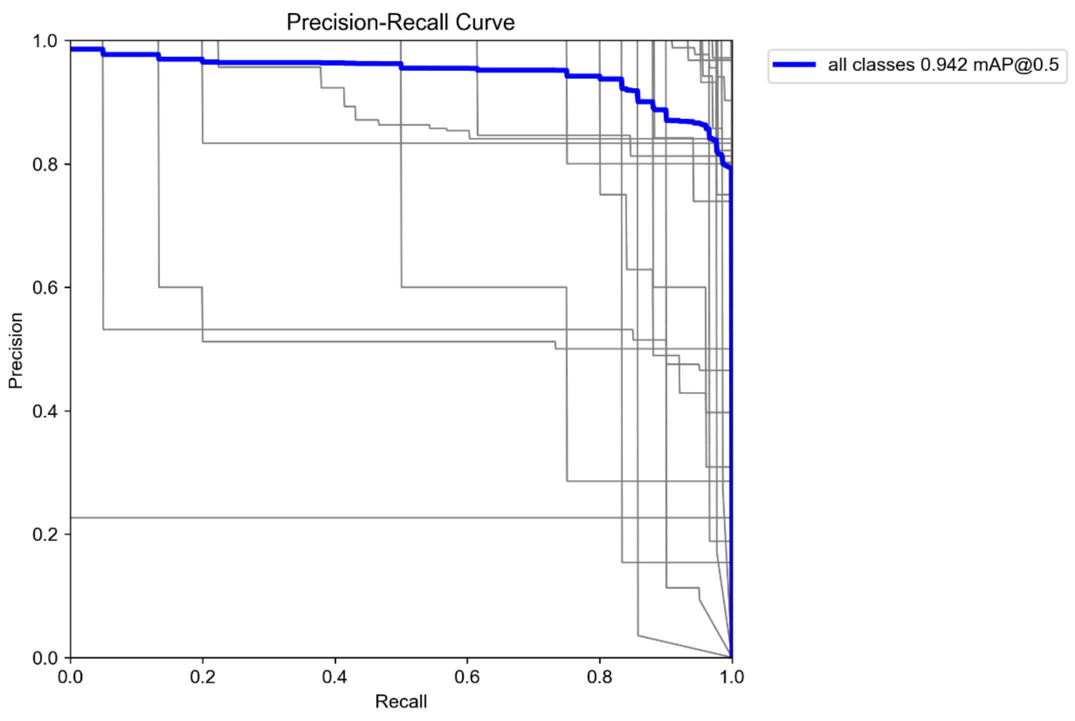


Figure 63: PR curve

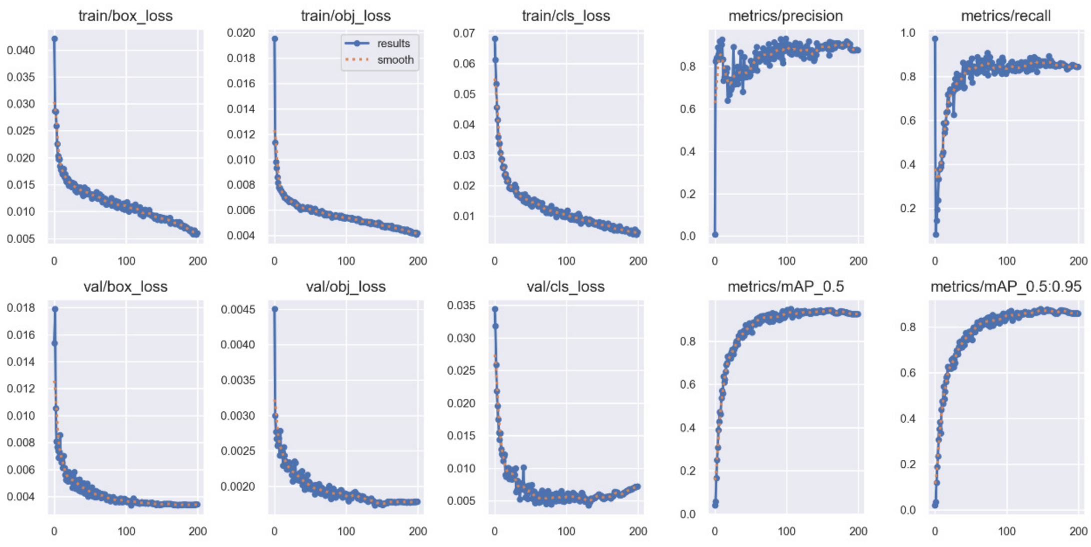


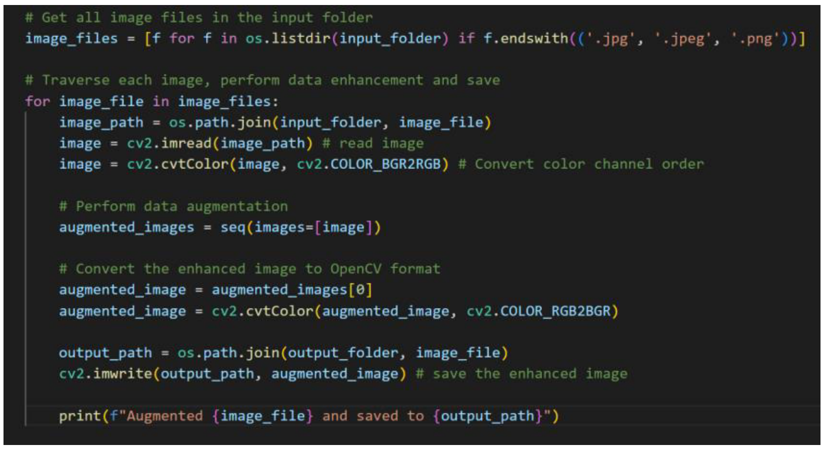
Figure 64: training result

It can be seen from the confusion matrix that the diagonal line is relatively obvious, indicating that the model as a whole can correctly classify most of the data. From the

PR curve, it can be concluded that the average mAP of all classes in the model is 0.942, but the mAP of each class looks very messy. This phenomenon is mainly caused by the uneven number of images in each class in the training set.

4.4.5 Data Augmentation

First, edit the data augmentation script aug.py. This part can traverse all the pictures in the folder and use the enhanced algorithm on them.



```python
# Get all image files in the input folder
image_files = [f for f in os.listdir(input_folder) if f.endswith(('.jpg', '.jpeg', '.png'))]

# Traverse each image, perform data enhancement and save
for image_file in image_files:
    image_path = os.path.join(input_folder, image_file)
    image = cv2.imread(image_path) # read image
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert color channel order

    # Perform data augmentation
    augmented_images = seq(images=[image])

    # Convert the enhanced image to OpenCV format
    augmented_image = augmented_images[0]
    augmented_image = cv2.cvtColor(augmented_image, cv2.COLOR_RGB2BGR)

    output_path = os.path.join(output_folder, image_file)
    cv2.imwrite(output_path, augmented_image) # save the enhanced image

    print(f"Augmented {image_file} and saved to {output_path}")
```

Figure 65: Data Augmentation Script

Then add the enhanced function seq of Figure 22 to the script. Finally run the script to process the training set. An example of the processed training data is as follows:

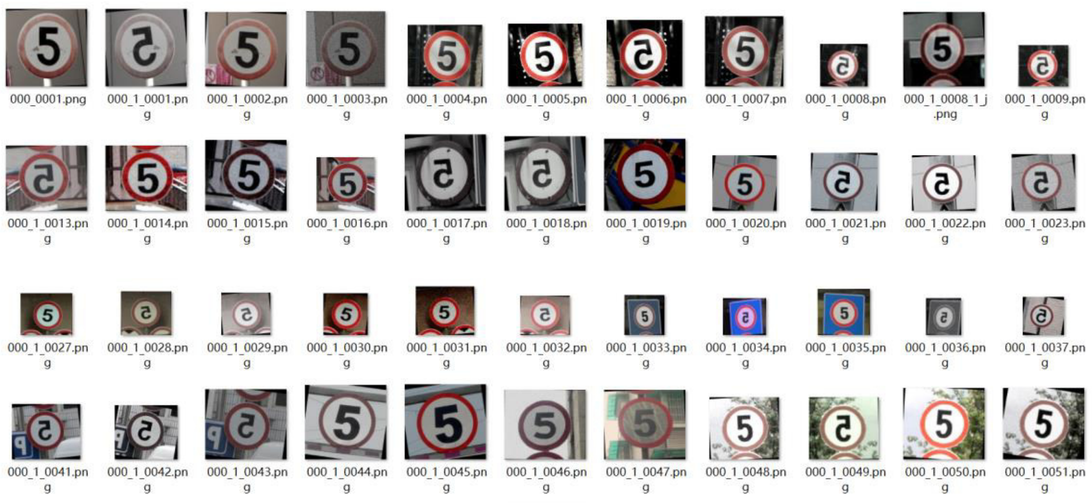


Figure 66: Dataset after data augmentation

It can be seen that the diversity and complexity of the processed data set have increased significantly. Replace the augment data into the training set directory and start training. The training results are shown below

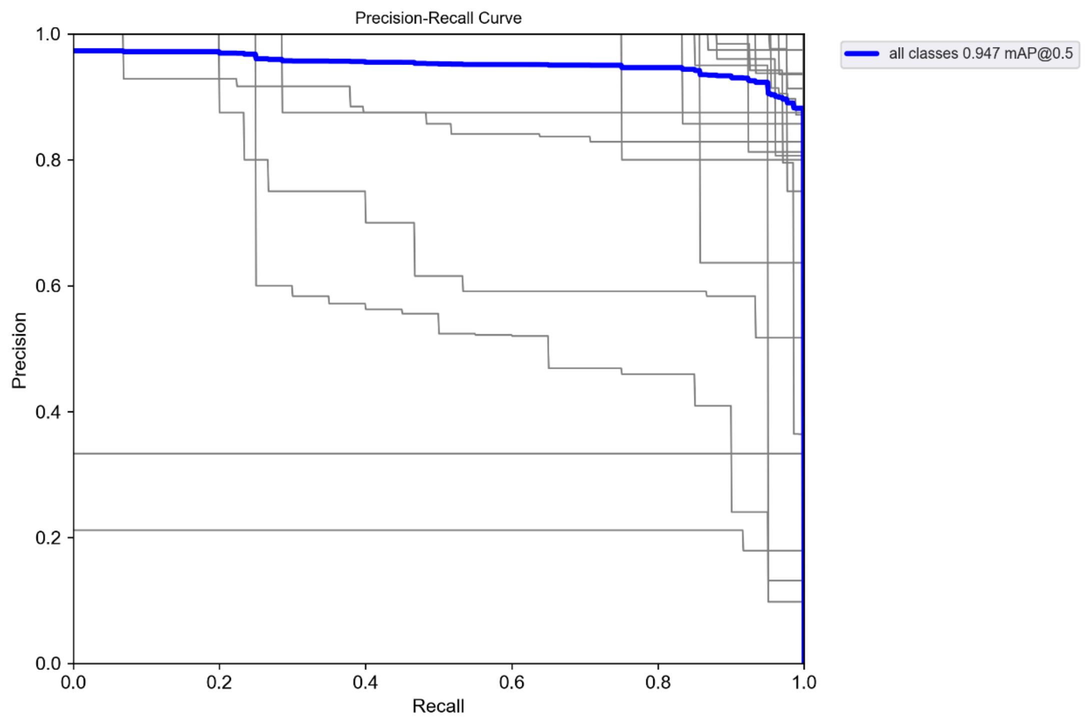


Figure 67: PR curve

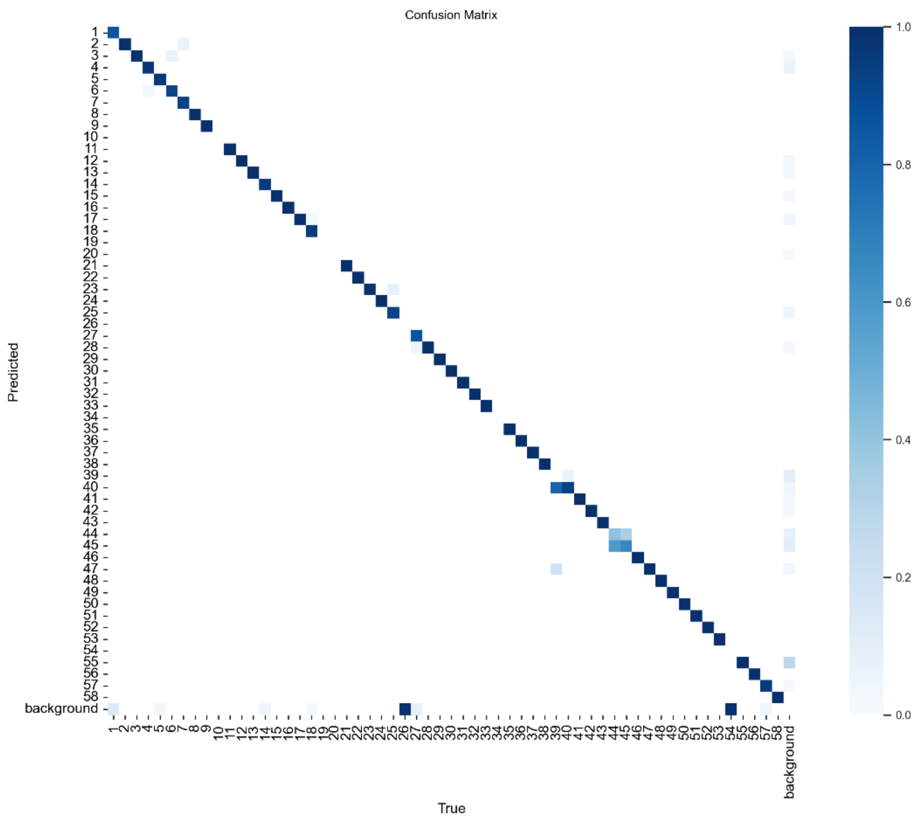
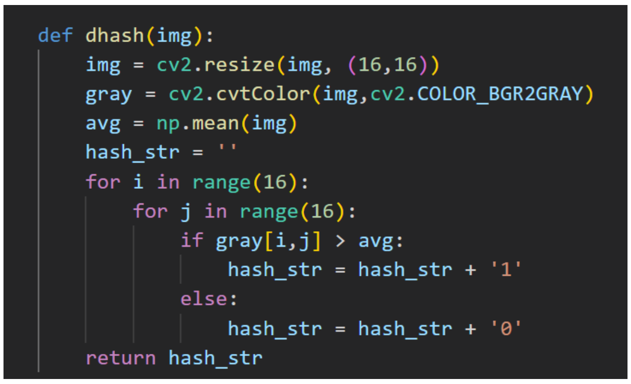


Figure 68: Confusion matrix

It can be seen from the PR curve that the model performs better on the enhanced data set, and the mAP has increased by 0.005. At the same time, it can be seen from the confusion matrix that the trained model can correctly classify most categories, and the performance has been greatly improved.

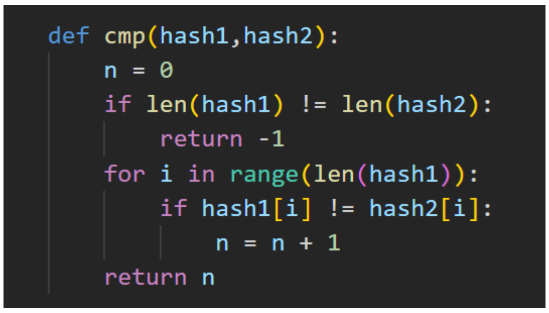
4.4.6 Dhash deduplication

First edit the deduplication script dhash.py. The main body is to generate hash string and comparison method.



```python
def dhash(img):
    img = cv2.resize(img, (16,16))
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    avg = np.mean(img)
    hash_str = ''
    for i in range(16):
        for j in range(16):
            if gray[i,j] > avg:
                hash_str = hash_str + '1'
            else:
                hash_str = hash_str + '0'
    return hash_str
```

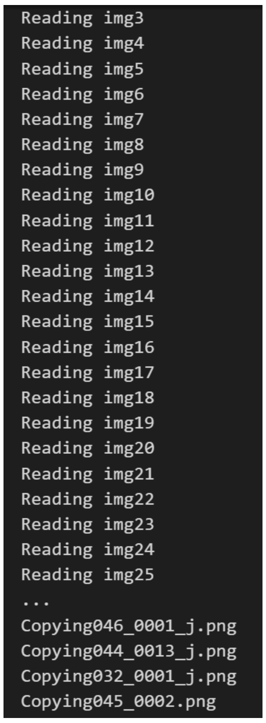
Figure 69: generate hash string



```python
def cmp(hash1,hash2):
    n = 0
    if len(hash1) != len(hash2):
        return -1
    for i in range(len(hash1)):
        if hash1[i] != hash2[i]:
            n = n + 1
    return n
```

Figure 70: comparison method

Then run the script against the tsrd dataset.



```
Reading img3
Reading img4
Reading img5
Reading img6
Reading img7
Reading img8
Reading img9
Reading img10
Reading img11
Reading img12
Reading img13
Reading img14
Reading img15
Reading img16
Reading img17
Reading img18
Reading img19
Reading img20
Reading img21
Reading img22
Reading img23
Reading img24
Reading img25
...
Copying046_0001_j.png
Copying044_0013_j.png
Copying032_0001_j.png
Copying045_0002.png
```

Figure 71: data processing

Here I set the threshold to 10. After processing, the images in the data set are reduced from 4170 to 1660. The data judged to be the same by the script is as follows:

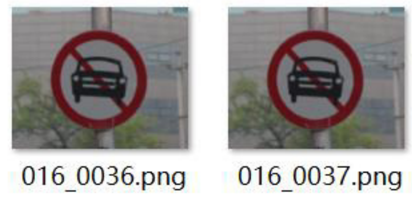


Figure 72: Deduplication data display 1

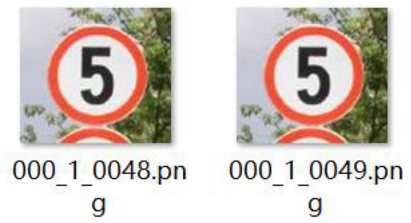
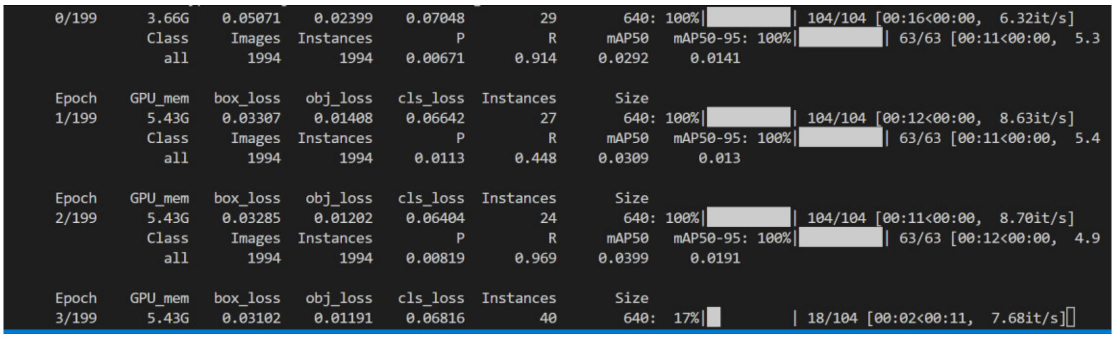


Figure 73: Deduplication data display 2

Finally attempt to use this dataset for training. The figure below shows a part of the epoch during the training process. It can be seen that due to the deduplication of the data set, the amount of data in the training set has been greatly reduced. Each epoch model will only call 104 pictures for learning.



```
  0/199      3.66G    0.05071    0.02399    0.07048         29        640: 100%|██████████| 104/104 [00:16<00:00,  6.32it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 63/63 [00:11<00:00,  5.3
               all       1994       1994    0.00671      0.914     0.0292     0.0141

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
     1/199      5.43G    0.03307    0.01408    0.06642         27        640: 100%|██████████| 104/104 [00:12<00:00,  8.63it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 63/63 [00:11<00:00,  5.4
               all       1994       1994     0.0113      0.448     0.0309      0.013

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
     2/199      5.43G    0.03285    0.01202    0.06404         24        640: 100%|██████████| 104/104 [00:11<00:00,  8.70it/s]
             Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 63/63 [00:12<00:00,  4.9
               all       1994       1994    0.00819      0.969     0.0399     0.0191

     Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
     3/199      5.43G    0.03102    0.01191    0.06816         40        640:  17%|█         | 18/104 [00:02<00:11,  7.68it/s]
```

Figure 74: Training process

Next observe the results.

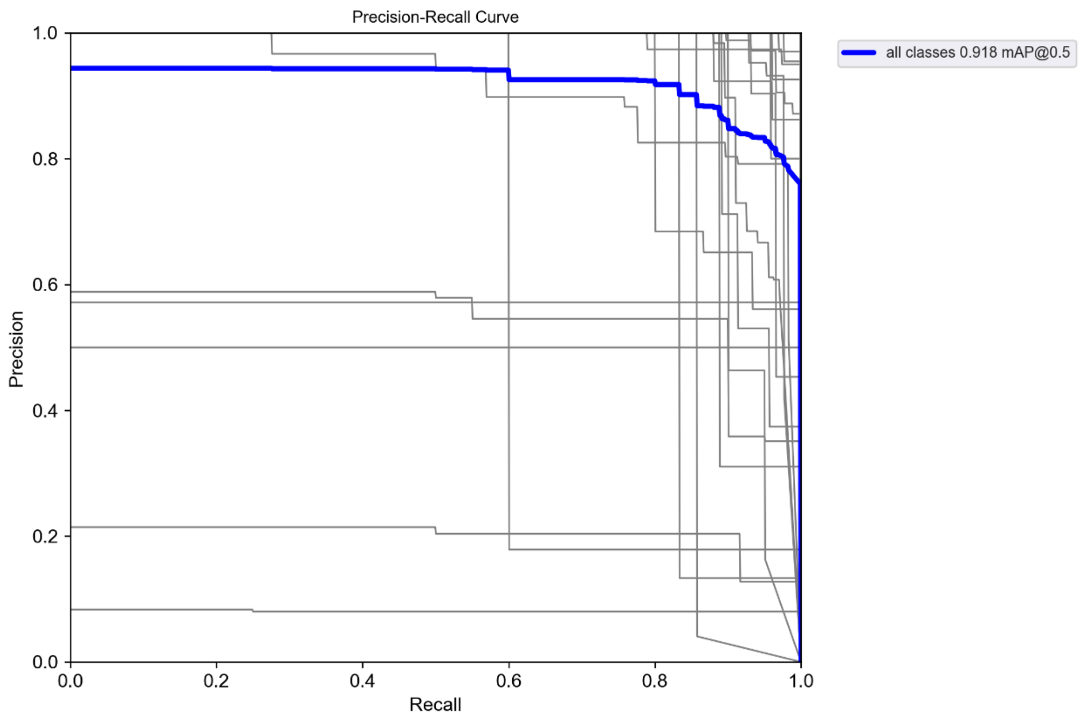


Figure 75: PR curve

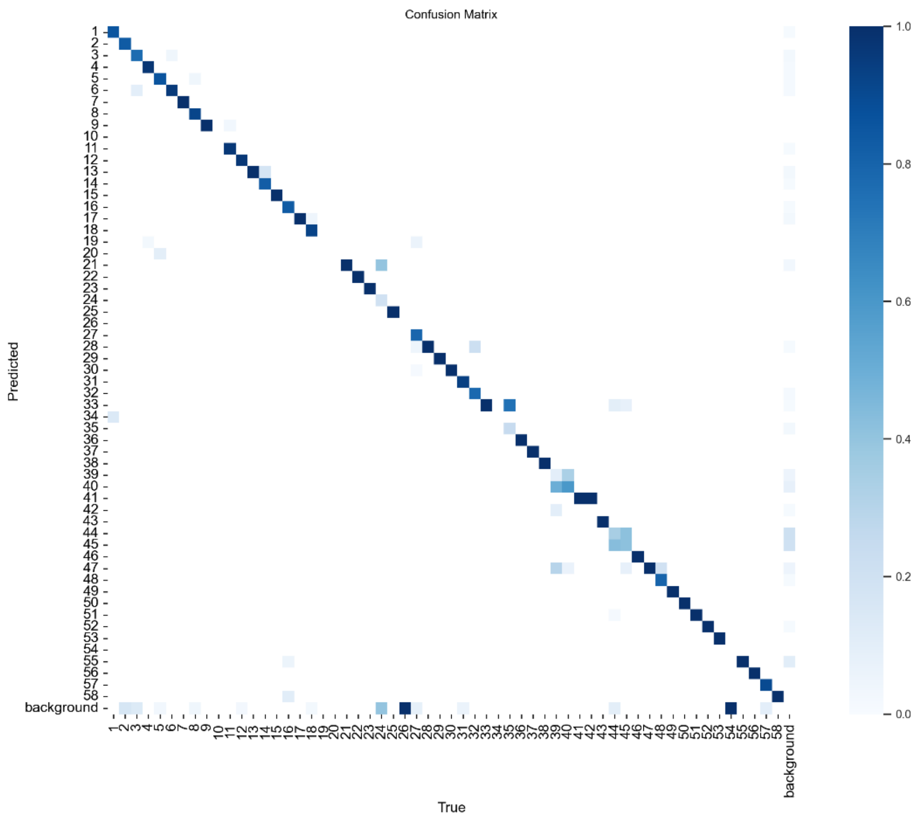


Figure 76: Confusion matrix

It can be seen from the PR curve that the deduplicated data set does not improve the mAP of the model. At the same time, it can be seen from the confusion matrix that the performance of the model in the background category is the same as the baseline, and many background objects are misjudged as correct objects. This result is not surprising. Because of in deep learning, the larger the amount of data, the better the training effect of the model is destined to be. Due to the relatively small data base of tsrd, the deduplicated data set cannot better support model training.

5 Compare and test

5.1 Object detection

In this project, I added SE block and CA block to yolov5s. Comparing them with the baseline respectively, it can be seen that the SE module cannot improve the model mAP, and the CA module can improve the smaller model mAP.

Table 5: Comparison of mAP of each training result

Baseline	0.785
Adding SE block	0.783
Adding CA block	0.791

From this result, it can be concluded that the SE module is not suitable for processing landmark object detection tasks. The main reason is that the SE module mainly uses the relationship between different channels in the image to add attention, and cannot add more effective attention to model by locating the specific position of the object. Since the proportion of road sign objects in the image is very small, it is often easy to ignore the important information, resulting in the inability to effectively improve the model's attention.

This is optimized in CA. CA attention is to use the object position coordinates to add attention to the model, so that the model can be sensitive to smaller objects.

Next, we test the results of the model. Here, five road sign pictures included in the test set classes are randomly selected from the Internet for testing. Here I use the model weight file with the highest mAP.

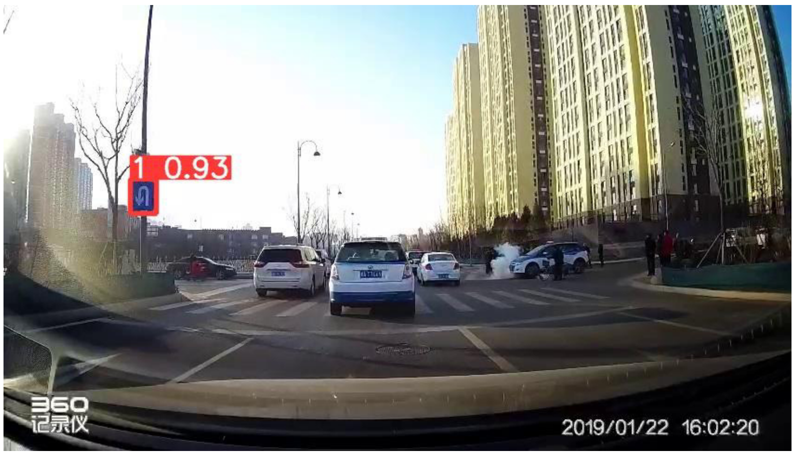


Figure 77: Test 1

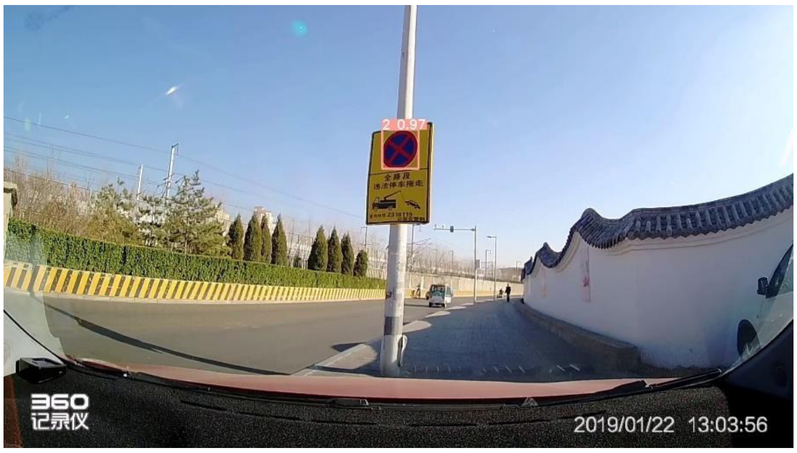


Figure 78: Test 2

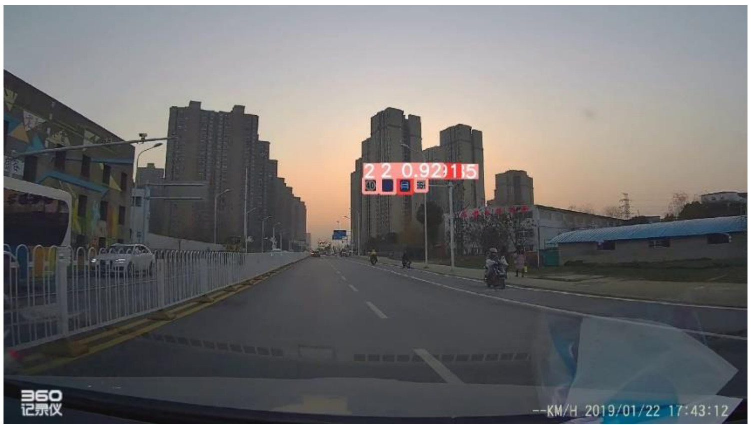


Figure 79: Test 3

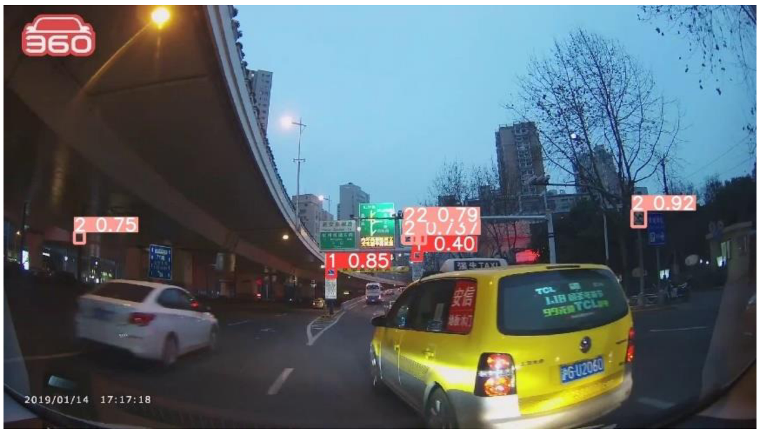


Figure 80: Test 4

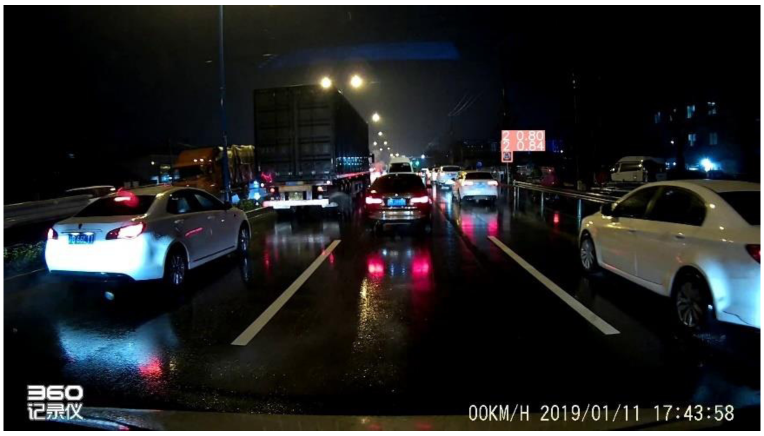


Figure 81: Test 5

It can be seen that even in a dim environment, the model can still correctly identify small road sign objects.

5.2 Object recognition

Here I summarize the mAP of baseline, data enhancement, and deduplication data.

Table 5: Comparison of mAP of each training result

Baseline	0.942
Data enhancement	0.947
Dhash deduplication	0.918

It can be seen from the table that data augmentation is effective in improving the performance of the model. Deduplication does not improve the performance of the model, but because many identical pictures are removed after deduplication, the speed of model training is significantly improved, and the speed is five times faster without losing a lot of mAP. But in any case, the mAP of the three is very high, indicating that yolo is relatively perfect for the recognition task. If you want a qualitative improvement, you may need to find a more powerful data enhancement method to enhance the training data, or use a more complete training set.

Next, we test the results of the model. Similarly, we choose five pictures of different categories for testing. Here I also use the model weight file with the highest mAP.

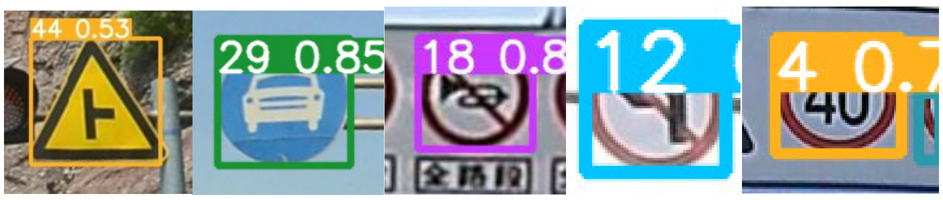


Figure 82: Test 1-5

From the results, it can be seen that the model can correctly identify and classify all objects and perform very well.

6 Conclusion

In this project, I used the yolov5 model to realize the two tasks of road sign detection and recognition. At the same time, in the detection task, I tried to add the attention mechanism SE and CA to the model. The comparison found that the CA module can effectively improve the mAP of the model; In the recognition task, I tried to use data enhancement to preprocess the model training data, and implemented the data enhancement algorithm by calling the imgaug library. It was found that data enhancement can effectively improve the model mAP. In the end, I called the trained weight file for recognition and detection tasks, and the results were all correct.

In the future work, I will continue the research in the field of machine vision and try to find more effective data augmentation methods, such as the SSD-based augmentation algorithm, which is an augmentation technique different from imgaug, which can change the coordinates of objects in the image position, and is able to generate many subgraphs for model training. Then I will try to use more and more complex data to train the model. Since this project only uses data based on Chinese road signs, there is still a lot of room for expansion of the data volume. In the end, I should try the tensorflow-based version of yolo to see if the framework can provide performance gains different from Pytorch, and if there is anything that can be improved.

7 Reflection on learning

Through this project, I have a deeper understanding of machine vision technology, and I also have a comprehensive understanding of YOLO, the most popular target detection model at present, and I am familiar with how to use it and improve it. At the beginning of the project, I didn't know the YOLO model, I only knew CNN and resnet. By consulting the Internet, I learned that the YOLO model is powerful and widely used in many industrial projects. However, due to the strong limitations of the target detection technology itself, different technologies are required to enhance the model due to the influence of different environments and factors. Based on this, I chose to use the YOLO model as the baseline and design an improvement plan to try to improve it. The result is satisfactory to me.

During the completion of this project, I also exposed many problems. The most important thing is that there are still many blind spots in my basic knowledge of machine learning. At the same time, I was still in the process of completing my graduation project, so I couldn't balance study and work well, resulting in very low efficiency and delays in work. This is one of the problems I most need to solve. In the future study and work, I should pay more attention to the learning of basic knowledge and improve the ability to handle multi-tasking at the same time.

The short year is coming to an end, and I will also graduate with a master's degree. I would also like to thank my alma mater, Cardiff University, for providing me with a high-quality learning environment and learning resources, and allowing me to meet many excellent teachers and classmates. At the same time, I would also like to thank the teachers of professional courses for their guidance, which enabled me to improve my knowledge and skills. In the future, I will continue to work and study in the field of artificial intelligence, and I will always remember this unforgettable time.

Reference

[1] McKinsey(2023). "The future of autonomous vehicles (AV)." [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected

[2] Jacob Solawetz(2020). "What is YOLOv5? A Guide for Beginners." (roboflow.com). [Online]. Available: What is YOLOv5? A Guide for Beginners. (roboflow.com)

[3] J. Zhang, X. Zou, L.-D. Kuang, J. Wang, R. S. Sherratt, and X. Yu, "CCTSDB 2021: A more comprehensive traffic sign detection benchmark," Human-centric Computing and Information Sciences, vol. 12, Article number: 23, 2022. DOI: 10.22967/HCIS.2022.12.023.

[4] Jianming Zhang, Wei Wang, Chaoquan Lu, Jin Wang, Arun Kumar Sangaiah. Lightweight deep network for traffic sign classification. Annals of Telecommunications, 2020, vol. 75, no. 7-8, pp. 369-379. DOI: 10.1007/s12243-019-00731-9.

[5] Jianming Zhang, Zhipeng Xie, Juan Sun, Xin Zou, Jin Wang. A cascaded R-CNN with multiscale attention and imbalanced samples for traffic sign detection. IEEE Access, 2020, vol. 8, pp. 29742-29754. DOI: 10.1109/ACCESS.2020.2972338.

[6]"Traffic Sign Recognition Database." (ia.ac.cn). [Online]. Available: http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html.

[7] "Unsupervised Feature Learning and Deep Learning Tutorial." (stanford.edu). [Online]. Available: http://ufldl.stanford.edu/tutorial/.

[8] AlexeyAB/darknet. "Repo Claims To Be YOLOv5 - Issue #5920." GitHub, [Online]. Available: https://github.com/AlexeyAB/darknet/issues/5920.

[9] ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite. GitHub, [Online]. Available: https://github.com/ultralytics/yolov5.

[10] ultralytics/yolov5. "YOLOv5 (6.0/6.1) brief summary - Issue #6998." GitHub, [Online]. Available: https://github.com/ultralytics/yolov5/issues/6998.

[11] Sambasivarao, K. "Non-maximum Suppression (NMS). A Technique to remove

duplicates and…" Towards Data Science. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[12] "Paperspace Blog. Mean Average Precision (mAP) Explained." [Online]. Available: https://www.paperspace.com/blog/mean-average-precision-mAP-explained.

[13] Squeeze-and-Excitation Networks Jie Hu[0000−0002−5150−1003] Li Shen[0000−0002−2283−4976] Samuel Albanie[0000−0001−9736−5134] Gang Sun[0000−0001−6913−6799] Enhua Wu[0000−0002−2174−1428]

[14] Q. Hou, D. Zhou, and J. Feng, "Coordinate Attention for Efficient Mobile Network Design," National University of Singapore, arXiv:2103.02907v1 [cs.CV], Mar. 4, 2021.

[15] "imgaug — imgaug 0.4.0 documentation," [Online]. Available: imgaug — imgaug 0.4.0 documentation.

[16] aleju/imgaug: Image augmentation for machine learning experiments. GitHub, [Online]. Available: https://github.com/aleju/imgaug.

[17] "The Hacker Factor Blog. Kind of Like That." [Online]. Available: Kind of Like That - The Hacker Factor Blog.

[18] "ultralytics/yolov5." GitHub, [Online]. Available: https://github.com/ultralytics/yolov5.

[19]"Chapter 5: The Retinal Representation," Foundations of Vision, Stanford University, https://foundationsofvision.stanford.edu/chapter-5-the-retinal-representation/.