TITLE: **YOLO Target Detection on The VOC 2012 Dataset**

SUBMISSON DATE: **March 19 2025**

Student ID	Name
24026719	YUANYUAN CHEN

Table of Contents

1. Introduction

In 2015, Jpesph and Redmon et al. first proposed a single neural network-based target monitoring system - YOLO. Subsequently, YOLO2 was proposed in CVPR (Computer Vision and Pattern Recognition) in 2017, which improved the accuracy and shortened the training time on the basis of YOLO1. Years of evolution followed. YOLO has been continuously updated with more versions, until March 2025, YOLO has been updated to the V12 version (Author(s), 2020). Yolo is an object detection model that inputs the input image into a neural network by normalizing the input image at a uniform size, and then performs multiple convolution operations on these features to finally get a feature map. The feature map is then divided into S×S grids, multiple bounding boxes are predicted according to each grid, and the one with higher confidence (the one with the highest confidence in this paper) is selected among all bounding boxes, and finally the detected target class and boundary are returned. He borrowed from GoogleNet, which has 24 convolutional layers and two fully connected layers. As the name suggests, YOLO (you only look once) allows you to predict what an object is and where it is (Redmon et al., 2016). Traditional image processing methods tend to slide the window to detect the image through the image pyramid to find the "interest" target of the model, which leads to a large number of calculations on the same image during training. The difference of Yolo is that YOLO performs global reasoning on the image when making a prediction, the whole image can be seen during training and testing, and the neural network can be run on the image to directly predict the detection result and then directly predict the target box, completing the target detection in one step (Redmon et al., 2016). Because we are going to detect the framework as a regression problem, we don't need a complex pipeline, so Yolo's very fast speed (Redmon et al., 2016). This feature makes YOLO more suitable for tasks dealing with large amounts of image data, such as video analysis and some tasks that require real-time feedback.
In this experiment, we used our self-trained target recognition model to classify 20 kinds of images in VOC_DATA dataset. It is mainly to predict the location of multiple targets (bounding box) and predict target categories.

2. Dataset

PASCAL VOC 2012 PASCAL VOC (Visual Object Classes) is a widely used data set in the field of computer vision, which is often used in object detection, semantic segmentation, image classification, and classical data sets. It was published by Pattern Analysis, Statistical Modeling, and Computational Learning in 2012 and is still in use today. The Pascal VOC 2012 dataset contains approximately 15,000 labeled images that fall into 20 categories. Categories are some common objects such as 'cat', 'dog', 'car', etc. (Shetty, 2012). The reason why VOC is selected as the dataset is that its image data is very rich, and a variety of scenes and categories make it more generalization ability to use it to train the model. Besides, it provides detailed annotation information and supports the visual model construction of many kinds of computers. Not only that, its data set is manually annotated, so the data noise is less, so the trained model will perform better.

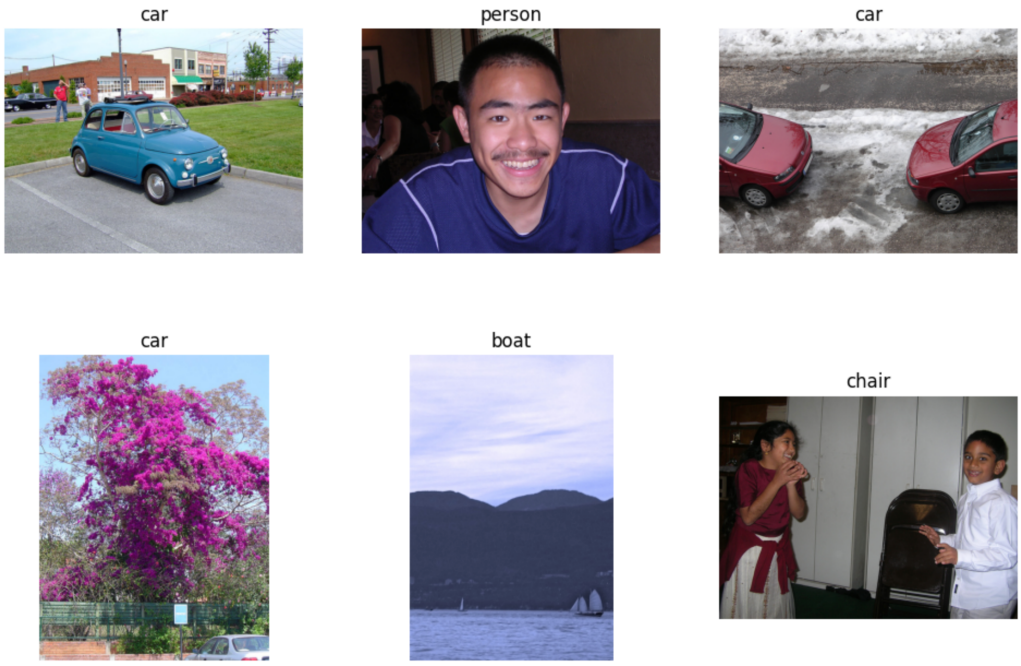


Figure 1 dataset preview

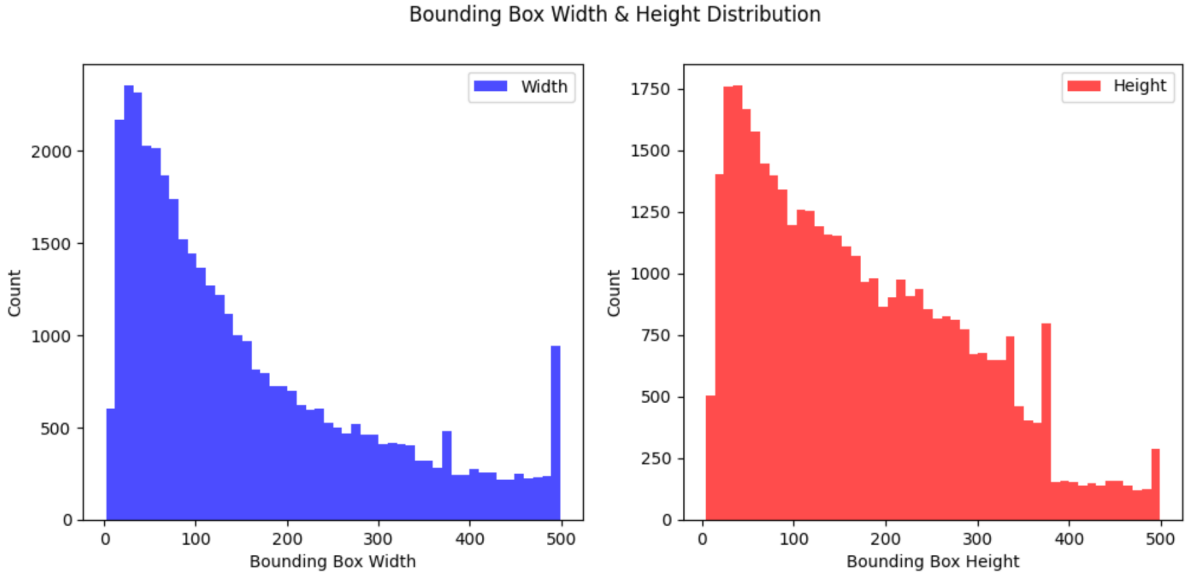


Figure 2 Bounding Box width and Height Distribution

3. Data preprocessing:

a. Normalize the Yolo tag by normalizing the coordinate values (x_center, y_center, width, height) to [0,1] to get the scale value of the image size, rather than the absolute pixel value. The VOC data set label file format needs to be converted into the yolo required file format, and the specific conversion is as follows:

	VOC data format	YOLOdata format
Mark file	XML	TXT
Coordinate format	Top left and Bottom right	Normalized center coordinates and width height
normalization	No normalization, use pixel coordinates directly	Coordinate values are normalized to [0,1] (relative to image size)

i. First, convert the file format from XML to TXT
ii. Second, calculate the position of the center point of the object and the width and height of the object box according to the upper left and lower right corners of the VOC
iii. Finally the coordinates are normalized to [0,1]
x_center = (xmin + xmax) / 2 / width
y_center = (ymin + ymax) / 2 / height
w = (xmax - xmin) / width
h = (ymax - ymin) / height

b. Image enhancement:
Fill and scale the image to make it square. Since the aspect ratio of the original image is not necessarily equal, in order to adapt the image to the input requirements of a network such as YOLOv1, we fill the image into a square and then scale the image to ensure that the size is 448x448
c. Image screening and division: Obtain all image files, keep only JPEG and PNG format pictures, then randomly select 300 pictures, and finally divide 90% training and 10% validation

4. Train

To avoid training from the beginning every time, we first pre-trained 500 images from the dataset, trained 20 epochs, and saved the trained model weights to the checkpoints folder. This way, in subsequent training, we can directly load the pre-trained weights without having to retrain the previous part of the model. Next, we selected 300 images, continued training using previously saved weight files, fine-tuning only the last few layers, and trained 5 epochs to get the final result. In this way, we significantly reduced training time while retaining the common features learned from pre-trained models, improving training efficiency and model accuracy. This model is trained in Google colab.

1. Parameters of training:

Parameters	**Value**
Num_workers	4

Batch_size	32
learning_rate	0.01
Epoches	25
Weight_decay	1e-4

2. Feature extraction of convolution

1) Convolution layer

Images entered in Yolo go through multiple convolutional layers to extract features that can then be used to predict the target boxes and categories for each grid region. ResNet18 is a deep convolutional neural network that has been trained on ImageNet (a large-scale image dataset) and learned to extract features, so it is particularly suitable for image classification tasks. We use ResNet18 as the backbone. And we removed the last two layers of the model (the fully connected layer and the pooled layer) , the previous convolution layer was retained, and some modifications were made to the convolution layer of ResNet18. This convolution converts the output from ResNet18 (512 channels) to 64 channels by convolution operation, with a convolution kernel size of 3x3 and padding=1 to keep the spatial size of the output unchanged. The 64 channels were then normalized in batches to help speed up training and improve stability. We use the LeakyReLU activation function to increase the non-linear expression of the network and also avoid "neuronal death". Setting to inplace=True can save memory. The next few convolutional layers continue to add depth and nonlinearity to the network.

```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        # ResNet18 as backbone
        resnet = tvmodel.resnet18(pretrained=True)
        resnet_out_channel = resnet.fc.in_features
        self.resnet = nn.Sequential(*list(resnet.children())[:-2])  # remove last 2 layer

        # convolution layer
        self.Conv_layers = nn.Sequential(
            nn.Conv2d(resnet_out_channel, 64, 3, padding=1),  # 输入通道数为 resnet_out_channel (5
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(64, 64, 3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(64, 128, 3, padding=1),  # output 128 = Fully connected input
            nn.BatchNorm2d(128),
            nn.LeakyReLU(inplace=True),
        )

        # Fully connected layer
        self.Conn_layers = nn.Sequential(
            nn.Linear(GL_NUMGRID * GL_NUMGRID * 128, 4096),  # The input dimension matches the c
            nn.LeakyReLU(inplace=True),
            nn.Linear(4096, GL_NUMGRID * GL_NUMGRID * (5 * GL_NUMBBOX + len(GL_CLASSES))),
            nn.Sigmoid()
        )
```

Figure 3 convolution layer

2）Flatten

Flattens the output of the convolution layer above so that the input is to the fully

connected layer. change to (batch, channel*height*width) change to (batch, channel*height*width)

```
class Flatten(nn.Module):
    """
    (n,c,h,w) -> (n, c*h*w)
    """
    def __init__(self):
        super(Flatten, self).__init__()

    def forward(self, x):
        n_samples = x.shape[0]
        x = x.reshape(n_samples, -1)
        return x
```

Figure 3 Flatten

3）Fully connected layer

This model has 2 fully connected layers, the input dimension of the first layer linear transformation is the feature graph size of the convolutional layer output (GL_NUMGRID * GL_NUMGRID * 128), and then the fully connected layer turns it into 4096. The second layer is the output layer, the output size is W*H*(5*B+C), which means that each grid contains 5 bounding box parameters, C is the number of categories, the following part will have a detailed introduction to the output

```
# Fully connected layer
self.Conn_layers = nn.Sequential(
    nn.Linear(GL_NUMGRID * GL_NUMGRID * 128, 4096),  # The input dimension matches th
    nn.LeakyReLU(inplace=True),
    nn.Linear(4096, GL_NUMGRID * GL_NUMGRID * (5 * GL_NUMBBOX + len(GL_CLASSES))),
    nn.Sigmoid()
)
```

Figure 4 Fully connected layer

4）The structure of the convolution network is shown in the figure below

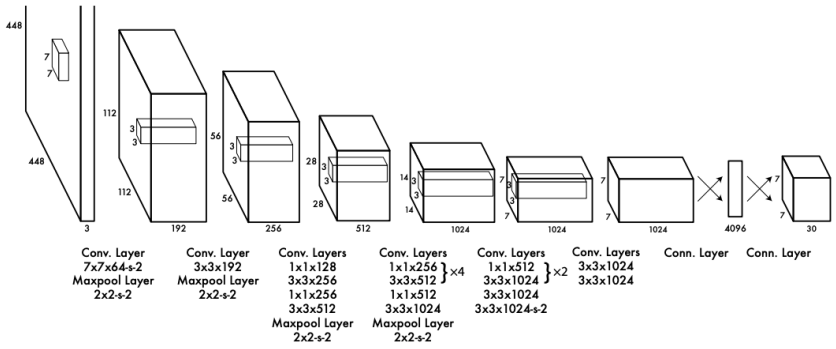


Figure 5 YOLO network structure taken from Redmon et al, 2015

5. Yolo Process

Yolo process mainly includes key steps such as grid partitioning, candidate box regression, confidence calculation, category prediction, and NMS (non-maximum suppression) processing.

1) Firstly divide S*S grids
2) Candidate box:

   In the YOLOv1 structure, each grid cell will predict 2 bounding boxes, which usually need to calculate some parameters to predict it. (x,y,w,h) is the prediction box

parameter of the network output. Confidence C means whether there is an object in the box, also need to calculate the Class probability, it represents the class of the object.

Predict the boundary of the target box (the position and size of the target), which is usually a regression task. The regression model parameters (tx,ty,tw,th) can be adjusted according to the following formula to adjust the position and size of the target box

$$\hat{x} = \sigma(t_x) + c_x$$

$$\hat{y} = \sigma(t_y) + c_y$$

$$\hat{w} = p_w e^{t_w}$$

$$\hat{h} = p_h e^{t_h}$$

Figure 6 Regression Formula

3) Target confidence

Each candidate box has an Object Confidence Score:

$$\text{Confidence Score} = P(\text{Object}) \times \text{IoU}(\text{Pred}, \text{GT})$$

P(Object) Represents the probability of whether the grid cell contains a target (binary classification task)

IoU is the Intersection over Union between the box and the Ground Truth.

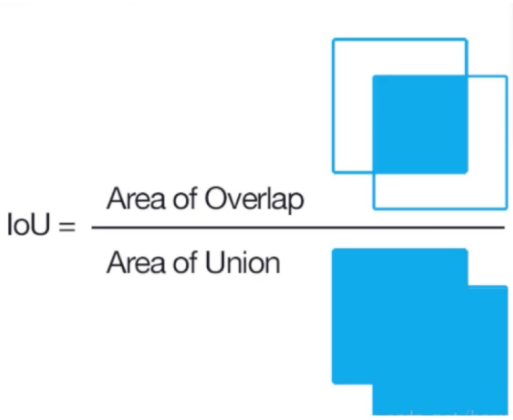


Figure 7 IoU

4) Calculate the class probability

For each grid, YOLO calculates the category probability that the target belongs to a certain grid: all boxes in each grid share the same set of category probabilities, and the confidence calculated in the previous step is multiplied with the category probability of the grid to obtain the category scores of different borders

$$\text{Class Score} = \text{Confidence Score} \times P(c_i|\text{Object})$$

Figure 8 class probability formula

5) Non-maximum Suppression (NMS)
Since YOLO may predict multiple overlapping target boxes, as shown in the figure below, non-maximum suppression (NMS) is required to remove redundant boxes. The box with the highest confidence is usually selected based on the set confidence threshold. Then calculate the IoU of this box and the other boxes, if the IoU is high, then it is probably a redundant box, delete the box and set the confidence level to 0.

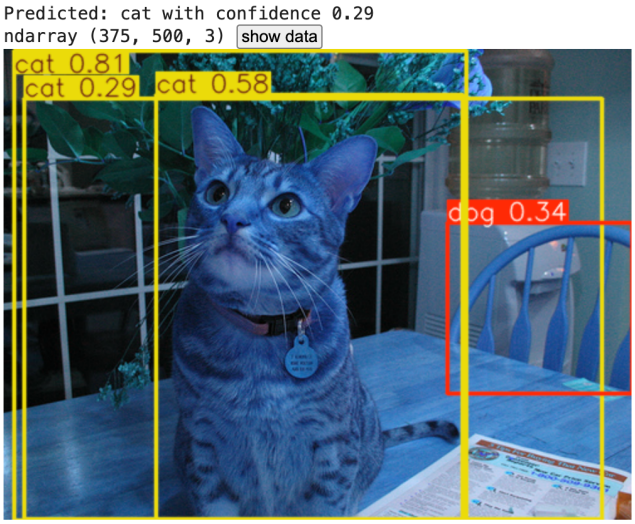


Figure 9 predict with multiple overlapping

6) Final output
   a. Each grid cell predicts two boxes: Each box will contain the following data:
   - 4 bounding box parameters (x, y, w, h)
   - 1 confidence (whether the target is included or not)
   - Classification probability of 20 categories
   - So the output for each box is 4 (box) + 1 (confidence) + 20 (class probabilities) = 25, which means each box has 25 values.

   b. Predictions for each grid cell: Each grid cell has two target boxes, each with 25 predicted values. So, for each grid cell, your output dimension is 2 * 25 = 50.
   Y = [Pc,bx,by,bh,bw,c1,c2,Pc,bx,by,bh,bw,c1,c2,…]

   c. Total output: Total grid: 3x3 = 9 grid cells. The output per grid cell is 50 (25 values for each of the two target boxes). So the total output is: 3 * 3 * 2 * 25 = 450.

6. Loss function

The loss function consists of three main components:

1) Bounding Box Loss (coor_loss)
   - The purpose of this parameter is to measure the coordinate error between the predicted box and the real box, make the predicted center coordinate (x,y) closer to the real target, and optimize the width and height (w,h) to better match the actual size of the target.
   - Since the width-height value may vary in a large range (the width-height value of a large target is larger), the error is calculated after taking the square root of the width-height in the function to reduce the error impact of the large box
   - Finally, this loss value is multiplied by 5 to give a higher weight, so as to focus more on target positioning.
2) Object Confidence Loss
   Confidence measures the likelihood that the forecast box contains the target, and its loss is divided into two parts:
   - obj_confi_loss: The confidence of the real object should be close to the IoU (the intersection of the prediction frame and the real frame). So we get the targeted confidence loss by calculating the mean square error between the forecast confidence and the IoU
   - noobj_confi_loss: The purpose is to calculate the confidence of the background region to prevent the model from mistakenly thinking of the background as a target. However, since the background area occupies the majority, in order to prevent excessive attention to the background and affect the target learning, multiply by 0.5 to reduce its weight
3) Classification loss: It is used to measure the error between the predicted category probability and the real category. The mean square error is used to calculate this difference, and then by minimizing the loss, the model can classify the target more accurately and improve the accuracy of category prediction.
4) The final loss calculation is to add coordinate loss, target confidence loss, target-free confidence loss and classification loss, and average the batch

Bounding box coordinate regression

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$

Bounding box score prediction

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

Classscore prediction

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{ classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

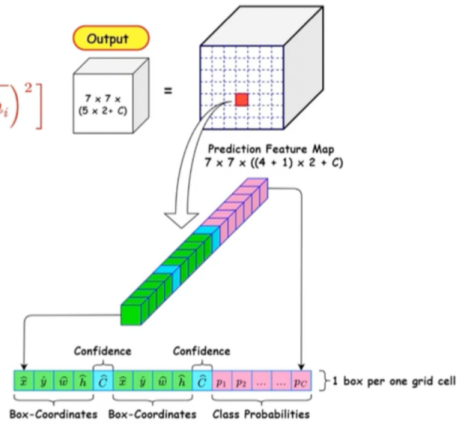


Figure 10 Loss functiopn formula

```python
def calculate_loss(self, labels):
    self.pred = self.pred.double()
    labels = labels.double()
    num_gridx, num_gridy = GL_NUMGRID, GL_NUMGRID
    noobj_confi_loss = 0.
    coor_loss = 0.
    obj_confi_loss = 0.
    class_loss = 0.
    n_batch = labels.size()[0]

    for i in range(n_batch):
        for n in range(num_gridx):
            for m in range(num_gridy):
                if labels[i, 4, m, n] == 1:  # if has object
                    # transfer(px,py,w,h)to(x1,y1,x2,y2)
                    bbox1_pred_xyxy = (
                        (self.pred[i, 0, m, n] + n) / num_gridx - self.pred[i, 2, m, n] / 2,
                        (self.pred[i, 1, m, n] + m) / num_gridy - self.pred[i, 3, m, n] / 2,
                        (self.pred[i, 0, m, n] + n) / num_gridx + self.pred[i, 2, m, n] / 2,
                        (self.pred[i, 1, m, n] + m) / num_gridy + self.pred[i, 3, m, n] / 2
                    )
                    bbox2_pred_xyxy = (
                        (self.pred[i, 5, m, n] + n) / num_gridx - self.pred[i, 7, m, n] / 2,
                        (self.pred[i, 6, m, n] + m) / num_gridy - self.pred[i, 8, m, n] / 2,
                        (self.pred[i, 5, m, n] + n) / num_gridx + self.pred[i, 7, m, n] / 2,
                        (self.pred[i, 6, m, n] + m) / num_gridy + self.pred[i, 8, m, n] / 2
                    )
                    bbox_gt_xyxy = (
                        (labels[i, 0, m, n] + n) / num_gridx - labels[i, 2, m, n] / 2,
                        (labels[i, 1, m, n] + m) / num_gridy - labels[i, 3, m, n] / 2,
                        (labels[i, 0, m, n] + n) / num_gridx + labels[i, 2, m, n] / 2,
                        (labels[i, 1, m, n] + m) / num_gridy + labels[i, 3, m, n] / 2
                    )
                    iou1 = calculate_iou(bbox1_pred_xyxy, bbox_gt_xyxy)
                    iou2 = calculate_iou(bbox2_pred_xyxy, bbox_gt_xyxy)

                    if iou1 >= iou2:
                        coor_loss = coor_loss + 5 * (torch.sum((self.pred[i, 0:2, m, n] - labels[i, 0:2, m, n]) ** 2) \
                                    + torch.sum((self.pred[i, 2:4, m, n].sqrt() - labels[i, 2:4, m, n].sqrt()) ** 2))
                        obj_confi_loss = obj_confi_loss + (self.pred[i, 4, m, n] - iou1) ** 2
                        noobj_confi_loss = noobj_confi_loss + 0.5 * ((self.pred[i, 9, m, n] - iou2) ** 2)
                    else:
                        coor_loss = coor_loss + 5 * (torch.sum((self.pred[i, 5:7, m, n] - labels[i, 5:7, m, n]) ** 2) \
                                    + torch.sum((self.pred[i, 7:9, m, n].sqrt() - labels[i, 7:9, m, n].sqrt()) ** 2))
                        obj_confi_loss = obj_confi_loss + (self.pred[i, 9, m, n] - iou2) ** 2
                        noobj_confi_loss = noobj_confi_loss + 0.5 * ((self.pred[i, 4, m, n] - iou1) ** 2)

                    # 检查类别标签是否有效
                    if labels[i, 10:, m, n].shape[0] > 0:
                        class_loss = class_loss + torch.sum((self.pred[i, 10:, m, n] - labels[i, 10:, m, n]) ** 2)
                else:
                    noobj_confi_loss = noobj_confi_loss + 0.5 * torch.sum(self.pred[i, [4, 9], m, n] ** 2)

    loss = coor_loss + obj_confi_loss + noobj_confi_loss + class_loss
    return loss / n_batch
```

Figure 11 Loss function code

7. Optimiser

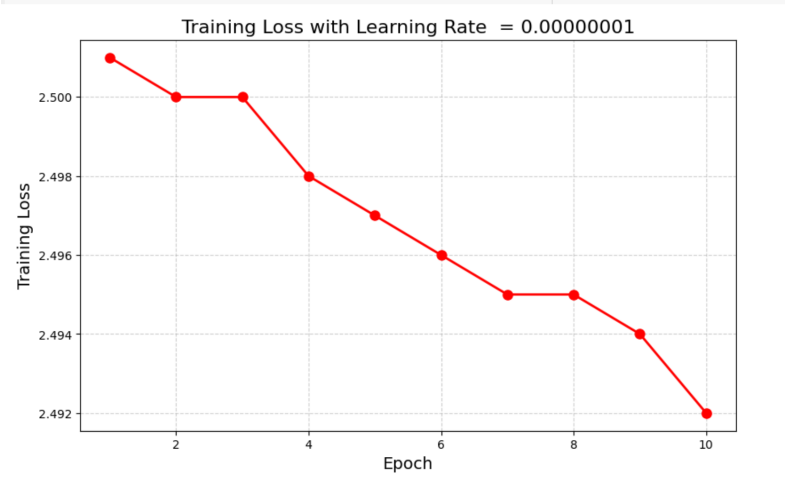
From the below results, we can see that although the loss value is very low, the accuracy of the model is still very low, which indicates that the loss function needs to be optimized. Cross entropy can be used to replace the mean square error (MSE) when calculating the classification loss, because the classification problem is usually multi-class one-hot coding, MSE is not sensitive to the difference in probability, resulting in the loss function does not reflect the characteristics of the classification problem well

8. Experiments

Adjustment of neural network architecture: At the beginning, the number of channels in each layer of the neural network we used was about 16-64 times as much as it is now, but when it was running, google colab directly crashed, or ran for a long time without any data output (crash picture).

After running, we tried to change the learning rate (from 0.001 to 1) to a maximum value to observe the change. Because the learning rate was too large, there was a gradient explosion. As shown below

Then we tried with the minimum (0.00000001), too slow gradient descent did not achieve good effects.



9. Result

1) The accuracy = 0.045 obtained is approximately 1/20, which is similar to the probability of random guessing, indicating that the model has not been trained enough.

Accuracy	0.0454
Precision	0.3056
Recall	0.0454

2) visualization

Due to the poor hardware conditions of our training, the result is very poor. The Loss value, IoU, and confusion metric are shown below to represent the result of this training

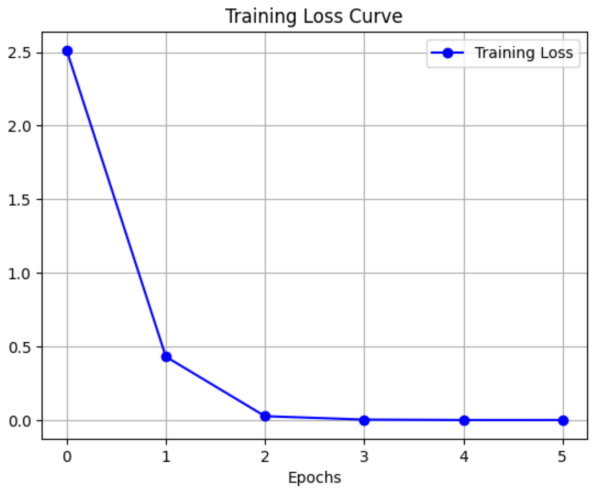


Figure 12 Loss value curve

The loss value of the curve is high at the beginning, and then decreases rapidly, especially in the first to second rounds, which indicates that the model starts to learn features, but because the data value is small, the model quickly learns the features of the existing data, resulting in the generalization ability of the modeler. So when the loss value is close to 0.002, the curve becomes relatively stable, indicating that the model has converged. And it goes into overfitting.

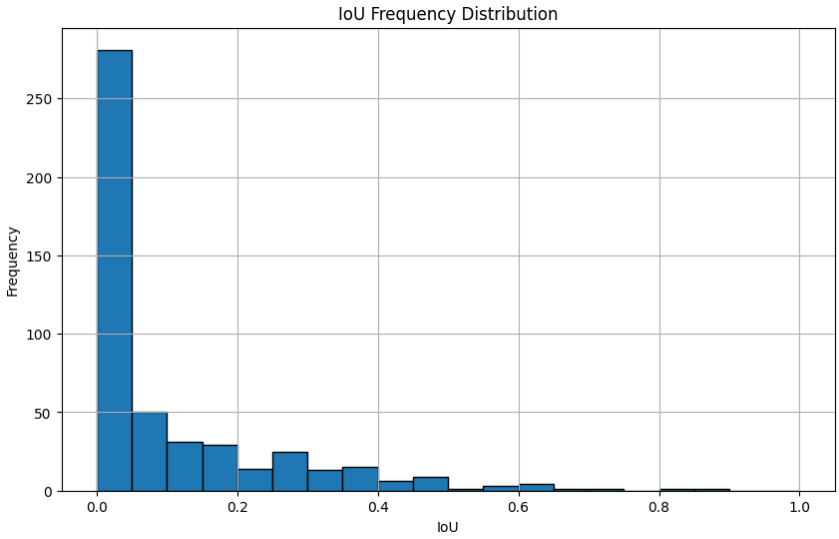


Figure 13 Iou and Frequency

It can be seen from the figure that there are many IoU equal to 0. This is because many boundary boxes are forced to be drawn and estimated, but the correct position is not predicted. The confidence of these boundary boxes is generally small, so the

reference value is not high. On the contrary, fewer boundary boxes with higher confidence have certain reference values.

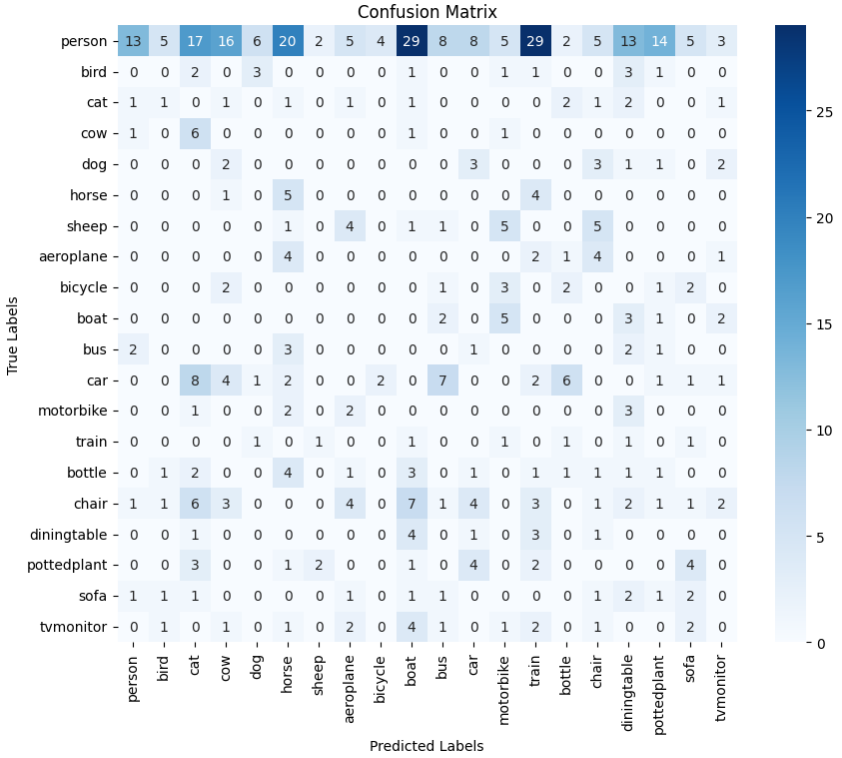
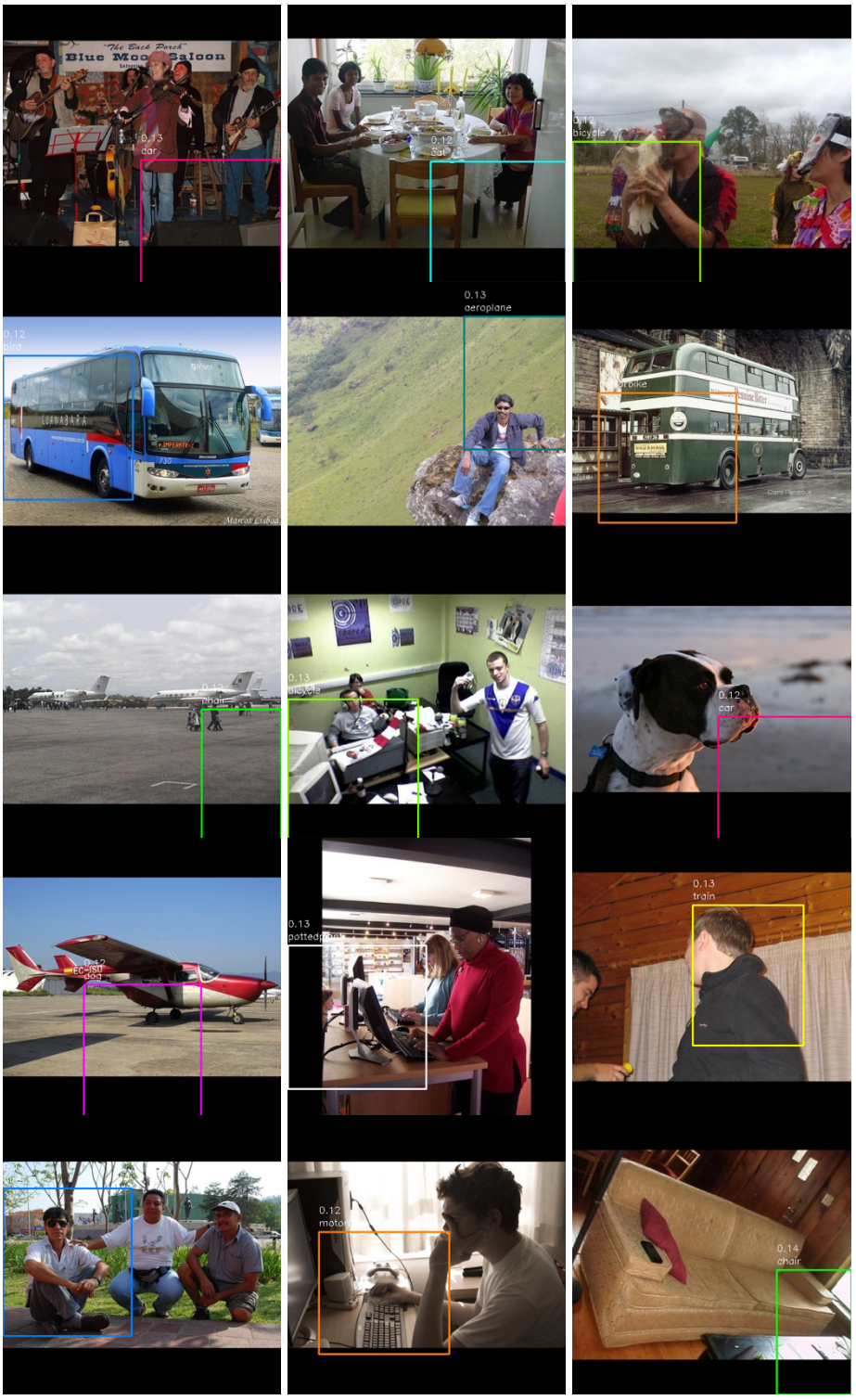


Figure 14 Confusion Matrix

From the confusion matrix, we can see that the data on the diagonal is scarce, so the accuracy of the model is not high, and it needs to be improved and trained with more images to achieve a certain accuracy. In addition, a large number of people are detected as ships in the predicted model, which indicates that the two have certain similarities in some features and need more data for training

3) Image visualization of YOLO prediction


0.13
car
0.12
cat
0.13
bicycle
0.13
aeroplane
0.12
bird
0.13
chair
0.13
bicycle
0.12
car
0.13
train
0.13
0.12
0.14
chair

The above is the boundry box detected by the model. Unfortunately, the data predicted by the model basically did not detect the target object and label it correctly. This shows that the generalization ability of the model is very weak, and it has not learned enough feature values of the target, so it is not enough to support the model to make accurate predictions.

10. Future Improvement

1. Due to equipment limitations, the width and height of the neural network are very small (most of the networks in this paper only have 64 channels in each layer), so I believe that the number of channels should be expanded to 16 to 38 times of the current level in order to reach the normal network level if possible in the future. In addition, we did not use the complete data set in the process of operation, training, including testing, but separated a small part of the data for training, which is limited to the equipment first, and also limited to the limited running time, so if there is a chance in the future, we should conduct a comprehensive upgrade of the entire model.

2. Upgrade for pre-training: Since YOLO has been built, the pre-training only uses different data to conduct a pre-training and store the model, and continues to use the previously trained weights in the subsequent training, which has some impact on the generalization ability of the model. For this, it is best to use a different model in the future, as well as another database (such as COCO) for some training, and then migrate the data to be more fully prepared for training the model.

3. For the visualization of the model, because the results we obtained are almost without any reference (the accuracy is very low), the output in the visualization process is not complete, so I hope that after training a model that can be used as a reference in the future, I will make all visual analysis completely.

4. epoch for training: In this experiment, because there are too few data that can be trained at one time, basically 3-4 generations after formal training, the loss function will completely converge, which is because there are too few data and too many kinds of overfitting. Therefore, in the future, more complex networks and more data support (complete VOC_2012 data) need the right number of epochs.2. From the results, we can see that although the loss value is very low, the accuracy of the model is still very low, which indicates that the loss function needs to be optimized. Cross entropy can be used to replace the mean square error (MSE) when calculating the classification loss, because the classification problem is usually multi-class one-hot coding, MSE is not sensitive to the difference in probability, resulting in the loss function does not reflect the characteristics of the classification problem well

11. Reference:

1. edmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016) 'You Only Look Once: Unified, Real-Time Object Detection', *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788. Available at: http://pjreddie.com/yolo/ (Accessed:15 march 2025).
2. Shetty, S. (2012) 'Application of Convolutional Neural Network for Image Classification on Pascal VOC Challenge 2012 dataset', *Manipal Institute of Technology*. Available at: [insert URL if available] (Accessed:15 March 2025).
3. Author(s) (2020) 'A Review of Yolo Algorithm Developments', *The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021)*. Available at: [insert URL if available] (Accessed:15 March 2025).
4. Weishuo (2024) 'YOLOv1 代码复现', *CSDN Blog*, 18 March. Available at: https://weishuo.blog.csdn.net/article/details/142643286 (Accessed: 15 March 2025).