MID TERM PROJECT FOR COMPUTER VISION 2022 SPRING

A PREPRINT

Keyu Mao
School of Data Science
Fudan University
220 Handan Road, Shanghai 200433
20307130241@fudan.edu.cn

Junhao Yuan
School of Data Science
Fudan University
220 Handan Road, Shanghai 200433
20307130129@fudan.edu.cn

Zehao Zhang
School of Data Science
Fudan University
220 Handan Road, Shanghai 200433
20307130201@fudan.edu.cn

May 14, 2022

ABSTRACT

In this paper we illustrate three computer vision tasks. We begin with the experiment of data augmentation on CIFAR-100 and comparisons among Cutout, Mixup and CutMix. Then we demonstrate the outline of object detection and elaborate the details of Faster RCNN and YOLOv3. Source code is available at https://github.com/mskmei/MIDTERM-PROJECT-CV-2022Spring.

Besides, pretrained models in this project are available at:

1. Data augumentation
   (1) PaddlePaddle model, extracting password: **2c4l**
   https://pan.baidu.com/s/1WksUUnOl8P2fgpLPFvJtzw
   (2) PyTorch model, extracting password: **huqt**
   https://pan.baidu.com/s/1UrhtZUk4bl4OztIQN44YrA
2. YOLOv3:
   https://drive.google.com/file/d/1Pglfi0Y8poLzsEPrNgKb0GYXpVjwKtn1/view?usp=sharing
3. Faster RCNN:
   (1) Resnet50-based:
   https://drive.google.com/file/d/1Ujds2mvsNLc8cXHH6827S-K$_Sfm V0Ssg/view?usp = sharing$
   (2) VGG16-based:
   https://drive.google.com/file/d/1niTR2cD5di8$_JfQyPEpgeyHfGGyCiUr/view?usp = sharing$

Keywords Data Augmentation · Object Detection · Deep Learning

1 Introduction

Since the proposal of Alexnet(1) in 2014, deep learning is getting increasingly popular among researchers. Besides, many relative fields including computer vision have also seen a rapid development. In this project, we aim to take an insight into data augmentation, a vital method in computer vision especially when there is a lack of data, and object detection.

Data augmentation is a ubiquitous strategy in the field of deep learning, where we duplicate data to combat overfitting. The return is appreciable, and within a few lines of code the model is likely to possess better robustness and improved accuracy. The simple tricks Cutout(2), Mixup(3) and CutMix(4) had once pushed the envelope of accuracy on classification tasks without introducing more complicated network architecture. We will in this project dive into the Cutout, Mixup and CutMix and display the results of experiments on CIFAR-100(5) dataset.

In terms of object detection, The object detection algorithm is roughly divided into two stages: the traditional object detection algorithm and the object detection algorithm based on deep learning. The first stage was around 2000. Most of the methods proposed during this period were based on sliding windows and artificial feature extraction, which had the defects of high computational complexity and poor robustness in complex scenarios. Representative achievements include Viola-Jones detector, HOG pedestrian detector, etc.

The second stage is from 2014 to the present, starting with the R-CNN algorithm proposed in 2014. These algorithms use deep learning technology to automatically extract hidden features in input images, and classify and predict samples with higher accuracy. After R-CNN, there are many image object detection algorithms based on deep learning such as Fast R-CNN(6), Faster R-CNN(7), SPPNet(8), and YOLO series(9)(10).

In this project,we will try to train two famous deep learning model, Faster RCNN and YOLOv3, on the VOC dataset(see details in the next part). These 2 models use different structures as feature extractor and have differences in dealing with feature maps. In the following parts, we will introduce these 2 models more specifically and compare them after making inferences about the same pictures.

2 Dataset

2.1 CIFAR-100

CIFAR-100, introduced in (5), is a widely used dataset in the field of computer vision. It contains $60,000$ natural images categorized in 100 different classes, $50,000$ of which are treated as training data, while the rest of which are for testing. Each of the image is in the size of 32×32 in RGB format. Commonly employed metrics include the top-1 accuracy and the top-5 accuracy. The state-of-the-art work reported in (11) has led the top-1 accuracy to 93.51%.

2.2 VOC2007&2012

The PASCAL VOC dataset(12) is one of the most commonly used standard datasets in the field of target detection. It can be divided into 4 categories and 20 subcategories in terms of categories. The category information is shown in the figure 1. The official of PASCAL VOC published a series of datasets from 2005 to 2012, among which VOC2007 and VOC2009 turned to be more well-known. That's because VOC2007 and VOC 2012 are mutually exclusive. VOC2012 includes all the images from VOC2008 to VOC2011 but has nothing to do with VOC2007. Therefore, Training and testing with these two datasets has been universally acknowledged as a superb approach.

In this project, we will use the training and validation datasets of VOC2007 as a new training set, use test dataset of VOC2007 as our new validation set, and finally use the validation dataset of VOC2012 as our new test set. Anyway, we must ensure that there will be no interation among our training, validaiton and test sets.The following table will visually show you the split of our train-validation-test set.

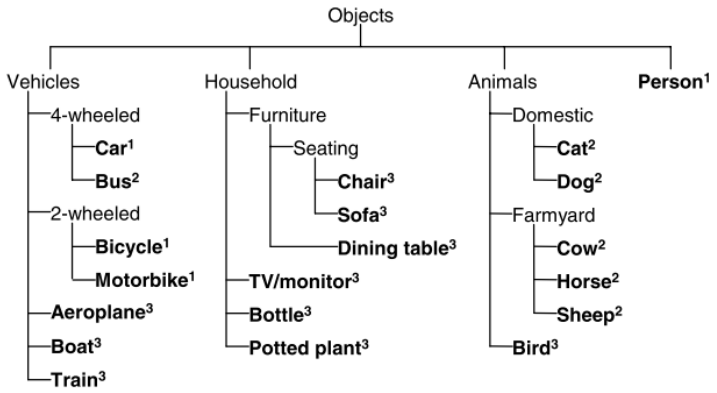


Fig. 2 VOC2007 Classes. Leaf nodes correspond to the 20 classes. The year of inclusion of each class in the challenge is indicated by superscripts: 2005[1], 2006[2], 2007[3]. The classes can be considered in a notional taxonomy, with successive challenges adding new branches (increasing the domain) and leaves (increasing detail).

Figure 1: PASCAL VOC Dataset

Our set(No interaction)	Training set	Validation set	Test set
Corresponding set	VOC2007 training set and validation set	VOC2007 test set	VOC2012 validation set

Table 1: The split of train-validation-test set based on VOC2007 and VOC2012

3 Data Augmentation on CIFAR-100

3.1 Principle

3.1.1 What is Data Augmentation

Data augmentation is one of the most widely-used tricks in training a neural network. As is often the case that data, especially those with labels, are limited whereas the world remains much to explore, models tend to overfit on the training data, resulting in poor generalization performance on testing ones.

To achieve better generalization performance and to increase the robustness of the model, previous works include weight regularizations and so on. In particular, He et al.(13) claims that training with more data is likely to be more effective than working with a pretrained network. With limited data, the idea of data augmentation lies in manually generating data and the associated labels.

In this paper, we are going to illustrate and reproduce some of the classical data augmentation tricks, including Cutout(2), Mixup(3) and the CutMix(4). We apply the ResNet(14) on the CIFAR-100(5) dataset for experiment. We will also analyze and compare the accuracy curve and the loss curve of different augmentation strategies.

3.1.2 Network architecture

We utilize the ResNet(14) as the backbone. The final linear layer has size 100 to match the number of classes in CIFAR-100. Displayed below is a demonstration of ResNet-18 borrowed from (15).

3.1.3 Loss function

Considering that there are soft labels in our experiment, we use the multilabel cross entropy, a generalization of the regular BCE loss. For example, in figure 3, as will be explicitly demonstrated shortly afterwards, there is a mixture of 42% house and 58% kangaroo and the label is set to 42% house plus 58% kangaroo accordingly. We recognize it

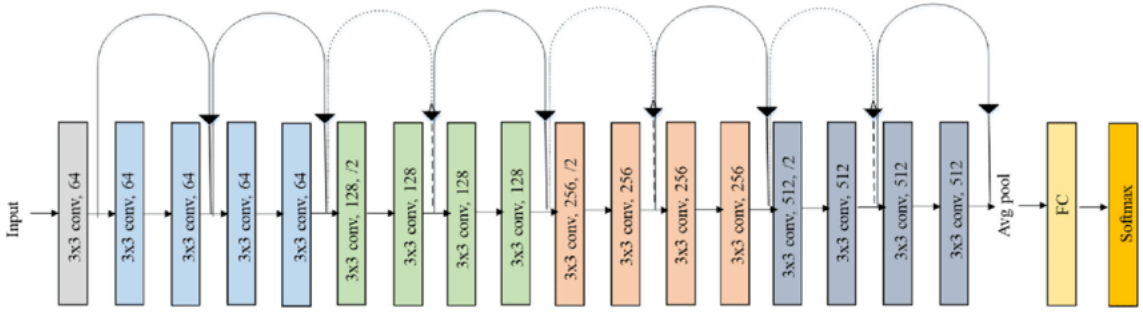


Figure 2: Caption

as the probability, that is, the image has a probability of 42% to be house while a kangaroo otherwise. Then the loss motivates to characterize the dissimilarity between probability distributions p, q in the form of

$$L = -\frac{1}{|C|}\sum_{k\in C}\left(q_k \log p_k + (1-q_k)\log(1-p_k)\right) \tag{1}$$

where C stands for the classes, p_k and q_k are predictions and labels for class k, respectively.

3.2 Data augmentation

3.2.1 Cutout

Cutout, proposed in (2), is one of the simplest methods for data augmentation that shares similar idea with cropping. Given a cutout size $H_1 \times W_1$ and the size of the image H, W, we uniformly sample a rectangular region $[h : h+H_1,\ w : w+W_1]$ on the image and replace it with black pixels as illustrated in the left column of 3.

As part of the information of the image gets masked in the process of Cutout, the label y is correspondingly updated, depending on the proportion of the region that has been cut out. It is formulated by

$$y \leftarrow \left(1 - \frac{H_1 W_1}{HW}\right) y \tag{2}$$

where H, W stands for the size of the image we are processing on.

3.2.2 Mixup

Mixup (3) is an alternating approach for augmentation. It innovatively blends two images and trains the network to predict two labels rather one. This is done by a linear interpolation between two images X_1 and X_2. Given α, we randomly sample $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$ and generate a new image X by pixelwise interpolation and the associated new label y is constructed similarly. Mathematically,

$$\begin{aligned} X &= \lambda X_1 + (1-\lambda)X_2 \\ y &= \lambda y_1 + (1-\lambda)y_2 \end{aligned} \tag{3}$$

where $y_!$ and y_2 are the onehot encoding of the labels for X_1 and X_2 respectively.

Mixup enables producing a great many of new images because there are billions of combinations. The labels learnt indicate that a mixed-up image is a linear combination of various categories. Hence it trains the network to learn the linear patterns behind to better generalize the model. However, as demonstrated in the middle of figure 3, a considerable number of images generated by Mixup are surreal and blurred, which can be actually hard for human eyes to distinguish.

3.2.3 CutMix

Now that we have learnt that Mixup generates seemingly unnatural samples that is intuitively likely to mislead the network. Also, as claimed in (4), the cutout augmentation suffers information loss and the deleted regions are somehow wasted in training. Then there comes the CutMix strategy, which overcomes both the problems by cutting out a region and replace it with another.

Formally, assume we are working with image X_1 of size (H, W). We begin with randomly picking up another image X_2 of the same size and sample from the Beta distribution a proportion parameter $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$. Here λ stands for the proportion that we mix X_1 with X_2. Take

$$H_1 = H\sqrt{1-\lambda} \qquad W_1 = W\sqrt{1-\lambda} \tag{4}$$

and we uniformly sample a submatrix $[h : h + H_1,\ w : w + W_1]$ where we replace X_1 by that of X_2. The proportion of update, given by $\frac{H_1 W_1}{HW} = 1 - \lambda$, indicates that X_1 and X_2 contributes to the resulting image with the proportion $\lambda : (1 - \lambda)$. This implies that the label should be updated correspondingly, i.e.

$$\begin{aligned} X_1[h : h + H_1,\ w : w + W_1] &\leftarrow X_2[h : h + H_1,\ w : w + W_1] \\ y_1 &\leftarrow \lambda y_1 + (1 - \lambda) y_2. \end{aligned} \tag{5}$$

In our implementation, we have followed setting $\alpha = 1$ as (4) did, where λ is sampled from a uniform distribution over $[0, 1]$.

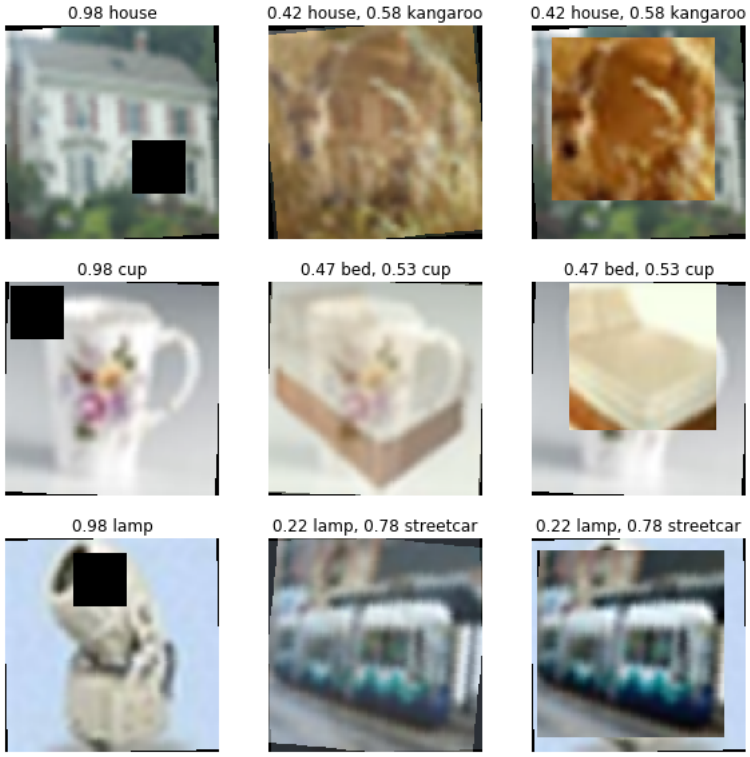


Figure 3: Data augmentation on three images from the training dataset, Cutout (left), Mixup (middle) and Cutmix (right) with random rotating or flipping as baselines.

3.3 Experiments

3.3.1 Settings

We have tested with non-pretrained ResNet-18(14). Some more hyperparameters are listed as below.

1. Batch size : 64
2. Initial learning rate : 0.003
3. Optimizer : Adam.
4. Weight decay (regularization): None.
5. Epoch and iteration: 140 epochs in total and 781 iterations per epoch.
6. Learning rate decay : Decay by 10x at 70 and 105 epochs.

3.3.2 Baseline and augmentation

Before Cutout, Mixup or CutMix, each CIFAR-100 image is firstly resized to $(224, 224)$ to match the standard input size of a ResNet, without which the accuracy will be significantly impacted by around 30%, possibly due to the aggressive downsampling in the head of a ResNet backbone. Then each image has probability 50% to be horizontally flipped and is rotated, either clockwise or counterclockwise, by an angle with no more than 15 degree.

For Cutout and CutMix, we use an area of 56×56, which is $\frac{1}{16}$ of a 224×224 image. For Mixup and CutMix demonstrated above, we set $\alpha = 1$ so that the distribution $\mathrm{Beta}(\alpha, \alpha)$ degenerates to a uniform distribution as (4) does.

3.3.3 Accuracy

In figure 6 we display a typical accuracy curve on training 140 epochs. Despite the baseline leads the highest in the very beginning, it suffers certain overfitting and the accuracy stops climbing since 10 epochs or so. Both Mixup and CutMix have surpassed the baseline after around the 15 epochs. CutMix indicates slight advantage over Mixup. Cutout, however, performs the worst in our environment setting with a noticeable drawback from the baseline. It also signals greater oscillation in the curve.

Table 2 further illustrates the average accuracy in multiple trials with 95% confidence intervals.

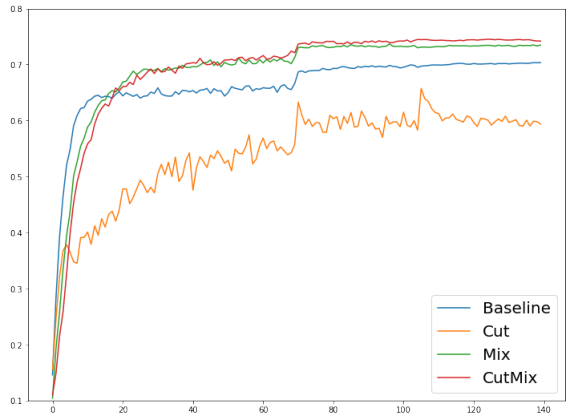


Figure 5: top-1 acc

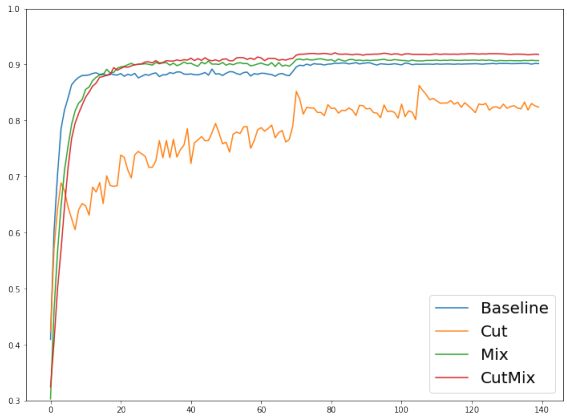


Figure 6: top-5 acc

Figure 6: Accuracy on testing dataset after training 140 epochs with learning rate decay by 10x at 70 and 105 epochs with different data augmentation. "Cut" stands for Cutout while "Mix" for Mixup.

Model	Pretrained	Top-1 Acc(%)	Top-5 Acc(%)
ResNet-18 + Baseline	×	68.06 ± 3.87	89.66 ± 1.48
ResNet-18 + Cutout	×	66.16 ± 2.44	86.79 ± 1.38
ResNet-18 + Mixup	×	70.70 ± 0.86	90.08 ± 0.55
ResNet-18 + CutMix	×	$\mathbf{72.63 \pm 3.59}$	$\mathbf{91.57 \pm 0.99}$

Table 2: Average accuracy with 95% confidence intervals on different augmentation schedules.

3.3.4 Loss

The training loss is plot in figure 9. The loss of baseline, Mixup and CutMix are displayed in the left. We observe that, the loss curve of the baseline, colored in blue, decreases most rapidly amongst the three. This implies that it has overfit early and provides small gradient after on. The loss curves of Mixup and CutMix share similar pattern and stable at approximately 1.4. But the latter is relatively smaller, corresponding to higher accuracy illustrated in figure 6.

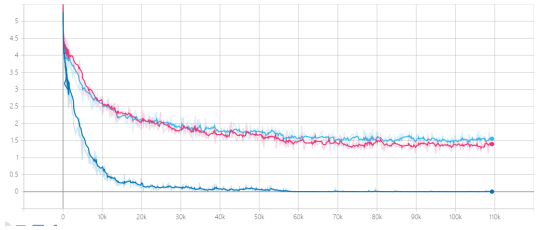


Figure 8: Baseline/Mixup/CutMix loss

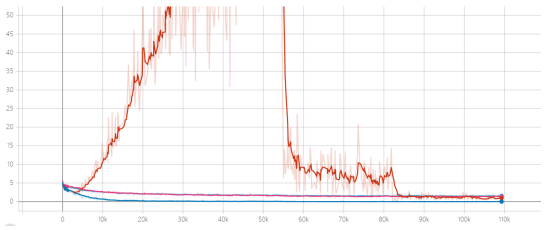


Figure 9: Cutout loss

Figure 9: Training loss recorded per 50 iterations. Left: Loss of the baseline (blue), Mixup (lightblue), CutMix (hotpink). Right: Explosive loss of Cutout (orangered) in comparison with others.

The loss function of Cutout is particularly displayed in the right. It shows an abnormal, explosive increasing in the 70 epochs before learning rate decay. The loss drops back after learning rate decay and gets smaller than those of Mixup and CutMix in the final epochs after number 105.

3.3.5 Conclusion

With our experiment setting, both CutMix and Mixup have significantly boosted the top-1 and top-5 accuracy on testing dataset. And CutMix outperforms Mixup by another 1.93% on top-1 accuracy. However, Cutout fails to rival the baseline, which we attribute to the unstability in information loss and the large learning rate. This suggests that Cutout require more careful learning rate selection or additional warmup strategy than Mixup and CutMix do. Overall, CutMix might always be a wise choice.

3.3.6 Future work

As far as we are concerned, the Cutout does not unleash its power as reported in (2) in the very experiment setting we choose. Hence, it remains to try out other hyperparameters, including optimizers, learning rates and weight decay. Also, one can also experiment with various backbones, for example ResNet-101 and VGG, and other datasets, It is also an interesting topic to test the generalization performance, characterized by how good the pretrained network are with these augmentation strategies on transfer learning in other computer vision tasks.

4 YOLOv3

4.1 Principle

4.1.1 What is YOLOv3

"You Only Look Once" or "YOLO" is a family of deep learning models which is designed for fast object detection. Until 2018, There are 3 main variations of YOLO, they were from version 1 to version 3. The fifth version named YOLOv5 was just launched by Ultralytics on GitHub last year.

The first version proposed the general architecture, while the second version improved the design and made full use of predefined anchor boxes to improve the bounding box proposal, and version three further improved the model architecture and training process.(16)

It is based on the idea that: A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.(9)Furthermore, similar to faster RCNN, YOLOv3 also use anchors to detect objects. However, instead of setting anchors manually, YOLOv3 use K-means to get proper anchors.

In the terms of extracting the characteristics of the picture information, compared with the region proposal feature extraction used in the FAST R-CNN target detection model, YOLOv3 chose to train the global region of the picture. When speeding up, it can better distinguish the target and the background, but for large objects. In other words, the background may also be counted as part of the target, so this is why it is particularly effective for small and medium objects, but the large background is not satisfactory.(17)

4.1.2 Net structure

YOLOv3 uses Darknet-53 as the backbone. That is, Darknet-53 is used as a feature extractor. Darknet-53 is mainly composed of 3 x 3 and 1 x 1 Convolutional with skip connections like the residual network in ResNet. Every convolutional consists of 3 parts: a convolutional 2D layer, a batch normalization layer and a Leaky ReLu layer(which is our activate function). (16)

The specific structure of the darknet can be seen in 10(a), and we also conclude the architecture of YOLOv3 in 10(b)(18) . Furthermore, YOLOv3 merely uses convolutional layers to downsample by setting stride as 2. In other words, YOLOv3 is a fully convolutional network without any use of pooling.(10)

In the experiment, for instance, the inputs is a batch of images of shape (batch size, 416, 416, 3).

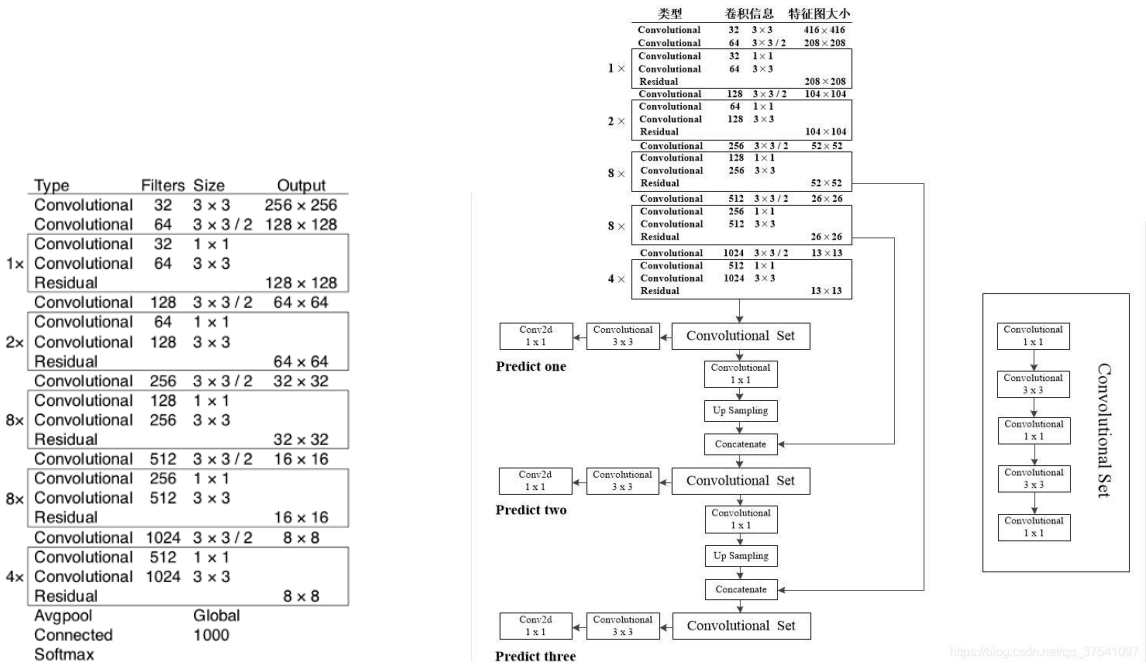
YOLO v3 will pass the image to a convolutional neural network. After the process of feature extraction, YOLO is able to generate 3 feature maps with different size. In our case, they are respectively 13x13, 26x26, 52x52. And then feature maps will pass (5+20)*3 1x1 kernel to predict the result, where 5 is (pc, bx, by, bh, bw), 20 is the number of classes and 3 is number of bounding box prior.

The size of feature maps determine the size of their receptive field and make their functions different. For instance, the 13x13 feature map has the largest receptive field on the original image, thus making it suitable for detecting large object.

Traditionally, researchers tend to use parameters pretrained on ImageNet. Nevertheless, in our experiment, **we use parameter pretrained on COCO dataset**. Anyway, pretrained parameters are used just in case that the model is initialized with terrible parameters and can't go through the training process smoothly.

4.2 Loss function

Due to the use of anchors, we should consider several factors when choosing loss function. In this experiment, box situation, confidence and classification turn to the three most important factors. Box situation can describe how far is



(a) Dark-net 53 (b) Yolov3

Figure 10: Net structure

a predict box of a positive sample from the corresponding true box. Confidence can intuitive show us how possible is it to have detectable objects. And classification is used to describe the accuracy of the prediction.

Considering those 3 factors, we plan to build the loss function with 3 parts: box_loss, obj_loss and cls_loss. On Github, there are sevral versions of loss function as the original paper didn't give a specific form of loss function(10). We finally choose the function given by Ultralytics, a research institute. This version has been proved to be useful, and even the latest version of YOLOv5(not official) refers to this function.

BCE(Binary Cross Entropy) function was mainly used to build loss function. It has a general form

$$L_{BCE}(y_i, p_i) = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i) \tag{6}$$

4.2.1 Box loss

First, as for box_loss function, we choose CIoU loss function. To show the formula of CIoU, we have to define several parameters. The penalty item R_{CIoU} was defined as:

$$R_{CIoU} = \frac{\rho^2(b, b_{gt})}{C^2} + \alpha v \tag{7}$$

where b and b_{gt} denote the center of predicted box and true box respectively, ρ denotes the euclidean distance between 2 points, and C denotes the diagonal distance of the smallest closure region that contains both the predicted and ground-truth boxes.α is the parameter used for trade-off and defined as:

$$\alpha = \frac{v}{(1 - IoU) + v} \tag{8}$$

and v is a parameter used to measure the consistency of aspect ratio. v is defined as:

$$\frac{4}{\pi^2}(arctan\frac{w_{gt}}{h_{gt}} - arctan\frac{w}{h})^2 \tag{9}$$

where w_{ht} and h_{gt} denote the width and height of true box respectively.

Therefore, the box_loss function denotes as:

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b_{gt})}{C^2} + \alpha v \tag{10}$$

4.2.2 Object loss(confidence loss)

In our experiment, we didn't use the same object loss as the original paper(10). Instead, we use BCE to calculate our object loss (or confidence loss).To be exactly, the object loss denotes as:

$$L_{obj} = \sum_{i \in grids} L_{BCE}(o_i, \hat{c_i}) = -\sum_{i \in grids} (o_i \log(\hat{ci}) + (1 - o_i) \log(1 - \hat{c_i})) \tag{11}$$

where $o_i \in [0, 1]$ denotes the CIoU(or you can set it as IoU or GIoU) of i^{th} prediction bounding box and true bounding box. $\hat{c_i}$denotes prediction confidence which is got from sigmoid function.

4.2.3 Classification loss

In terms of classification loss, we also didn't use the loss function in original paper(10). We also use BCE to calculate classification. The specific form is as follows:

$$L_{cls} = \sum_{i \in grids} \sum_{j \in classes} L_{BCE}(Y_{ij}, \hat{C_{ij}}) = -\sum_{i \in grids} \sum_{j \in classes} (Y_i j \log(\hat{C_{ij}}) + (1 - Y_{ij} \log(1 - \hat{C_{ij}}))) \tag{12}$$

$Y_{ij} = 1$ if objects of j^{th} class is truly in the i^{th} prediction bounding box. Otherwise $Y_{ij} = 0$. And $\hat{C_{ij}}$ denotes the predicted probability of the event that j^{th} class in the i^{th} prediction bounding box.

4.3 Total loss

The total loss is computed by those 3 types of loss above. We are supposed to give weight to each type of loss and add them together. The weight is not very important as we concentrate on the tendency of it. We just care about whether the training process is going smoothly. In this experiment, we the loss is computed as :

$$L_{total} = \lambda_1 L_{box} + \lambda_2 L_{obj} + \lambda_3 L_{cls} \tag{13}$$

where we set $\lambda_1 = \lambda_2 = \lambda_3 = 1$ by default. The loss figure of our training process and validation process can be seen in the part Metrics and visualization.

4.4 Optimizer and Learning rate adjustment strategy

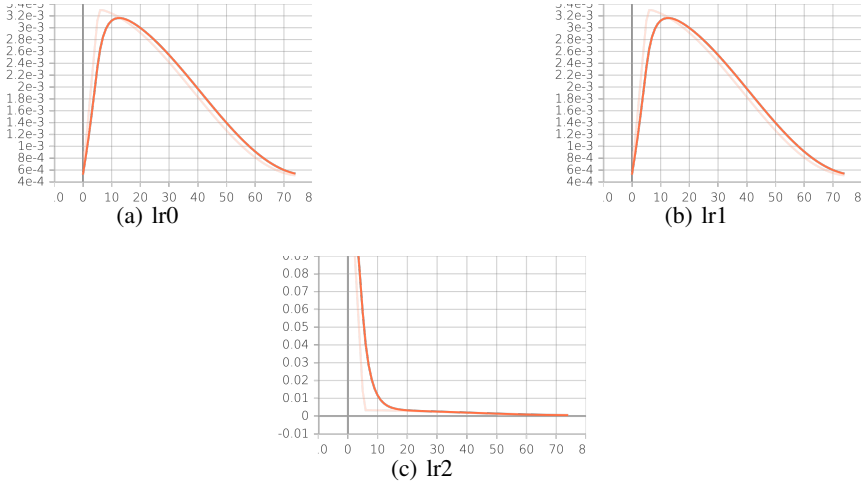
In our experiment, we use SGD as our Optimizer. And in order to adjust learning rates of BN layer, weight layer and bias layer respectively, we set three learing rates lr0(for BN layer),lr1(for weight layer),lr2(for bias layer).Furthermore, when adjusting BN layer, a momentum hyper parameter is added to improve the process, which use the gradient-based moving exponential weighted average to solve the problem of the large update amplitude swing of the mini-batch SGD optimization algorithm, and at the same time can make the network converge faster. Besides, we also add a l2 regularization when adjusting weight layer.

As for the adjustment strategy of learning rates, we first use warm-up method. lr0 and lr1 are set as 0.0 at first and will increase to initial learning rate(a fixed hyper parameter), while lr2 is set as 0.1 at first and will drop to initial learning rate. The number of epochs used for warm-up can be set before the experiment. After warm-up, we use Cosine Annealing method to update our learning rates, and the specific form is as follows:

$$lr_{new} = lr_{min} + \frac{1}{2}(lr_{initial} - lr_{min})(1 + \cos(\frac{T_{cur}}{T_{max}}\pi)) \tag{14}$$

where lr_{min} is a hyper parameter can contorl the minimum of learning rate and T_{max} is down cycle.

To show more details of the update process, we plot figures of three learning rates in the training process. You can see the update process of learning rates in Figure 11.



(a) lr0 (b) lr1

(c) lr2

Figure 11: Learning rate in training process

Images have been smoothed through tensorboard.

4.5 Experiment settings

In our experiment, we have several important hyper parameters which can be seen in the file hyp.VOC.yaml. We simply enumerate some of them:

1. Batch size : 32
2. Initial learning rate : 0.00334
3. Weight_decay(regularization): 0.00025
4. Epoch and iteration : 75 epochs, 157 iterations per epoch

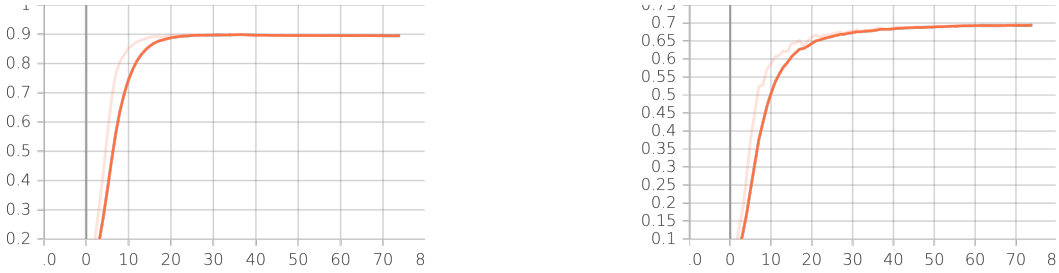
4.6 Metrics and visualization

In the experiment, we mainly focused on 3 metrics and the loss of both training and validation process. In the following subsection, we're going to visualize mAP(mean average precision),Accuracy and mIoU(mean interaction over union) in the validation process. The change of loss in both training process or validation process will be also given.

4.6.1 mAP@.5 and mAP@.5:0.95

mAP is an indispensable metric in the field of object detection, and it's the mean average precision of our validation process. To compute mAP, we should first get the average precision of each class and get the mean value of those classes.

In the experiment, mAP@.5 and mAP@.5:0.95 are used. The number behind "@" is IoU threshold. Detected samples with higher IoU(with groud truth box) will be regarded as positive samples. TO get mAP@.5:0.95, we are expected to set threshold from 0.5 to 0.95(the stride is 0.05) and compute mAP respectively, and then get the mean of all of those mAPs. We visualize the results of validation in Figure 12(a) and Figure 12(b).



(a) mAP@.5 (b) mAP@.5:0.95

Figure 12: mAP and mAP@5:0.95

Obviously, both mAP and mAP@.5:0.95 have an increasing tendency and finally reach a nice and reasonable value.The AP reached almost 0.9 and stayed steadily. As mAP is a decisive factor in object dectection, we can claim that the training and validation processes are successful.

4.6.2 Accuracy

In our experiment, accuracy is calculated by confusion matrix, which will be introduced specifically. Suppose M to be the confusion matrix, in this detection problem of multiple classes, we can simply get accuracy by:

$$Accuracy = \frac{\sum_{i \in class} M_{i,i}}{\sum_{i \in class} \sum_{j \in class} M_{i,j}} \tag{15}$$

We plot the tendency of validation accuracy in Figure 13

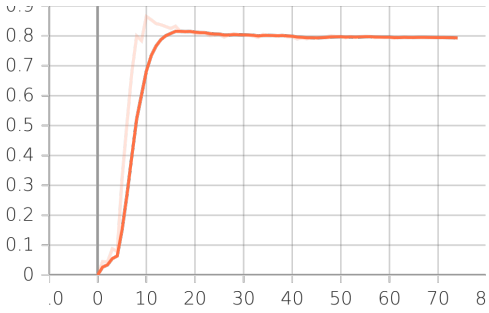


Figure 13: Accuracy of validation process

The accuracy of validation process has a obvious increase tendency. The accuracy increased rapidly in the first 20 epochs and remained steadily.

4.6.3 mIoU

Similar to the calcultion of accuracy, we use confusion matrix to compute mIoU. Confusion matrix provides us with a convenient way to get access to mIoU(19).

Before computing mIoU, we need give some definitions.First, "I" is defined as a vector which consists of the diagonal elements of confusion matrix."U" is defined as $actual_count + predict_count - I$, where

$$actual_count[i] = \sum_{j \in class} M_{j,i} \tag{16}$$

$$actual_count[i] = \sum_{j \in class} M_{i,j} \tag{17}$$

Then the mIoU can be calculated as

$$mIoU = mean(\frac{I}{U}) = \frac{\sum_{i \in class} I_i / U_i}{classes} \tag{18}$$

The Figure 14 is going to show you the tendecy of mIoU in the validation process.

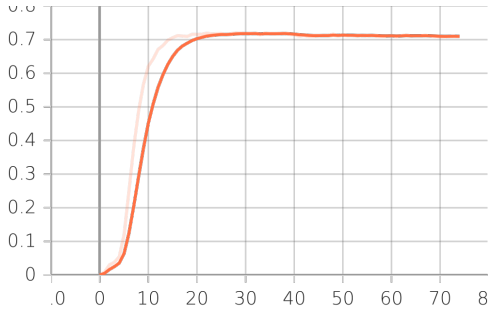
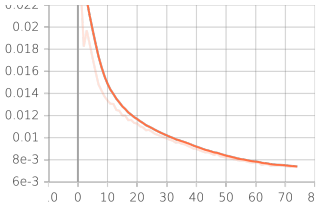


Figure 14: mIoU of validation process

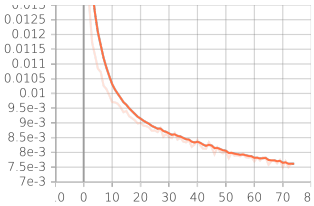
4.6.4 Loss

The specific details of loss have been introduces in 5.3. In this part, we simply give the visualization results.

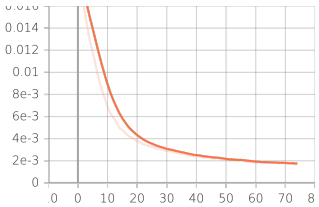
First, the following figure 15 are respectively box loss, object loss, classification loss and total loss of **training process**.



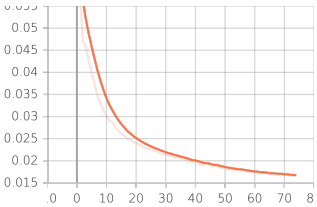
(a) box loss



(b) object loss



(c) classification loss



(d) total loss

Figure 15: The loss of **training process**

Then, the following figure 16 are respectively box loss, object loss, classification loss and total loss of **validation process**.

4.6.5 Loss

The specific details of loss have been introduces in 5.3. In this part, we simply give the visualization results.

First, the following figure 15 are respectively box loss, object loss, classification loss and total loss of **training process**.

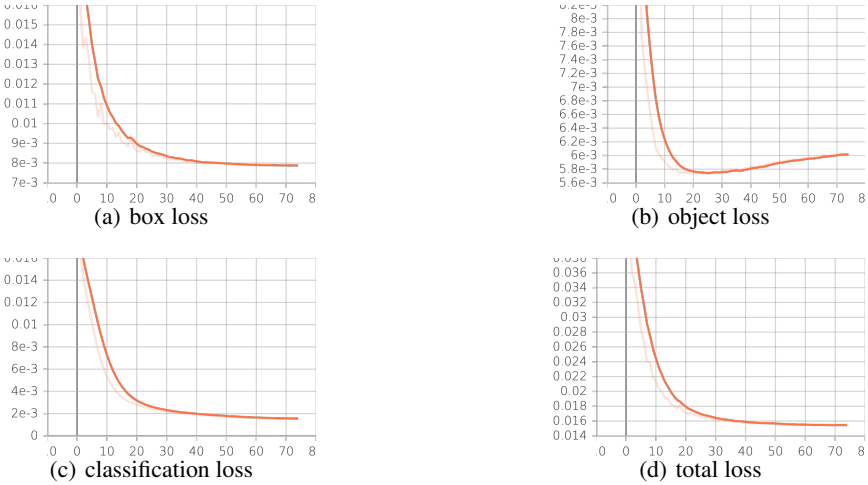


Figure 16: The loss of **validation process**

We simply find that almost all the loss have an decreasing tendency, which means the training and validation process go smoothly.The object loss have a slight tendency to rise after 30 epochs, but that didn't change the tendency of the total loss in validation process.

4.6.6 Results

We test out model with best weights on voc2012 validation dataset. We get the result as follows in figure 17.

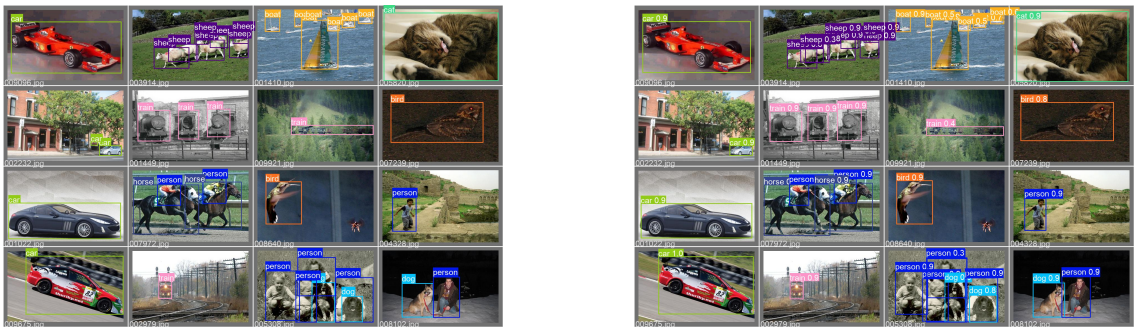
To simply visualize the effect of final model, we show half a batch of images(16 images) in the last epoch of validation process. In the figure 18 below, figure 18(a) is the true labels of each image while figure 18(b) is the predicted labels of each image.

4.7 Extra experiments

Besides required content, we also do some extra experiment. We will concentrate on precision-recall curve, confusion matrix and data augmentation in the following parts.

```
val: New cache created: ../datasets/VOC_2012/labels/val2012.cache
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 182/182 [(
                   all       5823      13841      0.863      0.788      0.838      0.658
             aeroplane       5823        433      0.943      0.883      0.907      0.748
               bicycle       5823        358       0.91      0.874      0.906      0.715
                  bird       5823        559      0.917      0.864      0.901      0.684
                  boat       5823        424       0.79      0.674      0.715      0.476
                bottle       5823        630       0.83      0.706      0.767      0.551
                   bus       5823        301      0.909      0.874        0.9      0.793
                   car       5823       1004      0.895      0.815      0.893      0.696
                   cat       5823        612      0.882      0.913      0.929      0.797
                 chair       5823       1176      0.814       0.58      0.675      0.487
                   cow       5823        298      0.855      0.832      0.879      0.698
           diningtable       5823        305      0.767      0.511      0.597       0.45
                   dog       5823        759      0.906      0.851      0.902      0.757
                 horse       5823        360      0.929      0.869      0.929      0.772
             motorbike       5823        356      0.908      0.857       0.92      0.721
                person       5823       4372      0.857      0.908      0.935      0.724
           pottedplant       5823        489      0.681       0.64      0.646      0.413
                 sheep       5823        413      0.855      0.841      0.873      0.704
                  sofa       5823        285      0.783        0.6      0.713      0.573
                 train       5823        315      0.929      0.872      0.905      0.752
             tvmonitor       5823        392      0.904      0.796      0.862      0.656
Speed: 0.1ms pre-process, 29.9ms inference, 1.7ms NMS per image at shape (32, 3, 416, 416)
```

Figure 17: test results



(a) True labels (b) Predicted scores and labels

Figure 18: The result of half batch in the last epoch of validation process

4.7.1 Precision-Recall curve

In the experiment above, we've already calculated the mAP of the validation process. However, we also want to distinguish different classes through AP(Average precision).

To visually show the differences among classes, we plot a P-R curve (precision-recall curve).Precision and recall are 2 common metrics used in object detection, which can be easily computed through confusion matrix.P-R curve use recall as xlabel and precision as ylabel.The area enclosed by the curve and the two axes is proportional to the AP(average precision) of each class. Therefore, we can use P-R curve(Figure 19) to describe AP of each class.

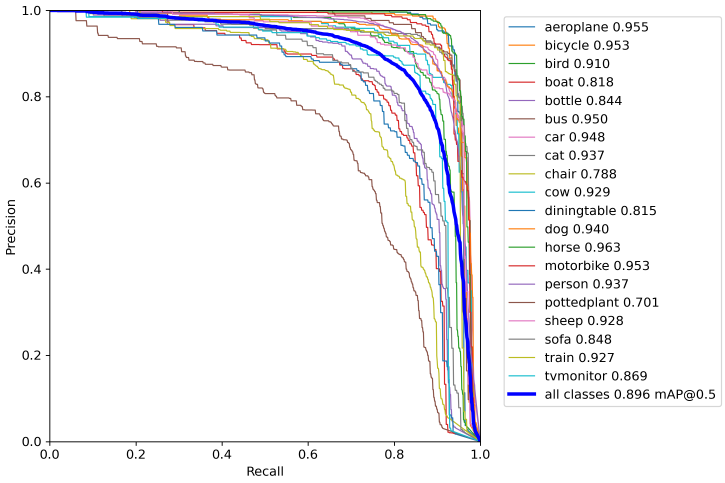


Figure 19: PR Curve

4.7.2 Confusion matrix of YOLOv3

The confusion matrix played an essential role in our experiment. As for confusion matrix, the vertical direction is true values of each class and horizontal direction is predicted value of each class. That is, only the place where the abscissa is equal to the ordinate is the place where the prediction is correct.In other words, only diagonal elements are true positive values.(thus we can compute accuracy easily)

The final confusion matrix of validation process is shown below in Figure 20.

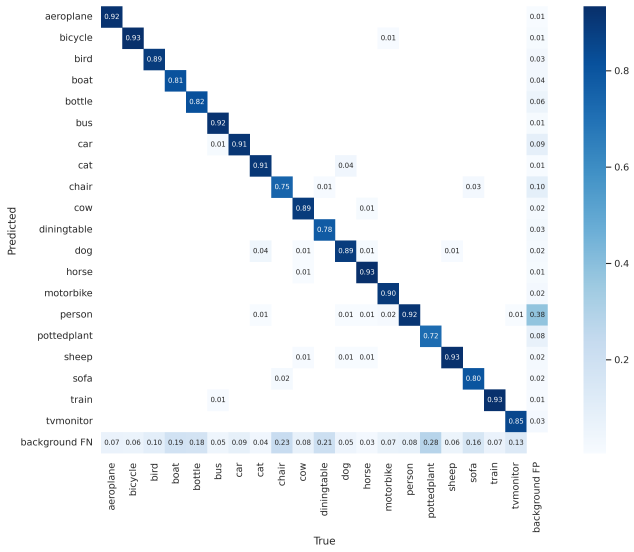


Figure 20: PR Curve

4.7.3 Data augmentation

In our experiment, we use a newly developed method Mosaic to do data augmentation.Actually, Mosaic method was proposed in YOLOv4(not official) instead of YOLOv3. However, this method can also be used in YOLOv3 and make sense.This part will give you a brief introduction to it.

In the article of YOLOv4, Mosaic data augmentation turns to be a new data augmentation method for mixing 4 images, which refers to CutMix's method of mixing only 2 input images(20). Its advantages: On the network model, due to hardware resources, the batch size cannot be set too large, and the batch size is indirectly increased by the Mosaic method.

It can be mainly divided into 3 parts:

1. Each time read 4 images from dataset
2. Flip, zoom, change the color gamut of the four pictures respectively, and place them in four directions.
3. Combine pictures and boxes

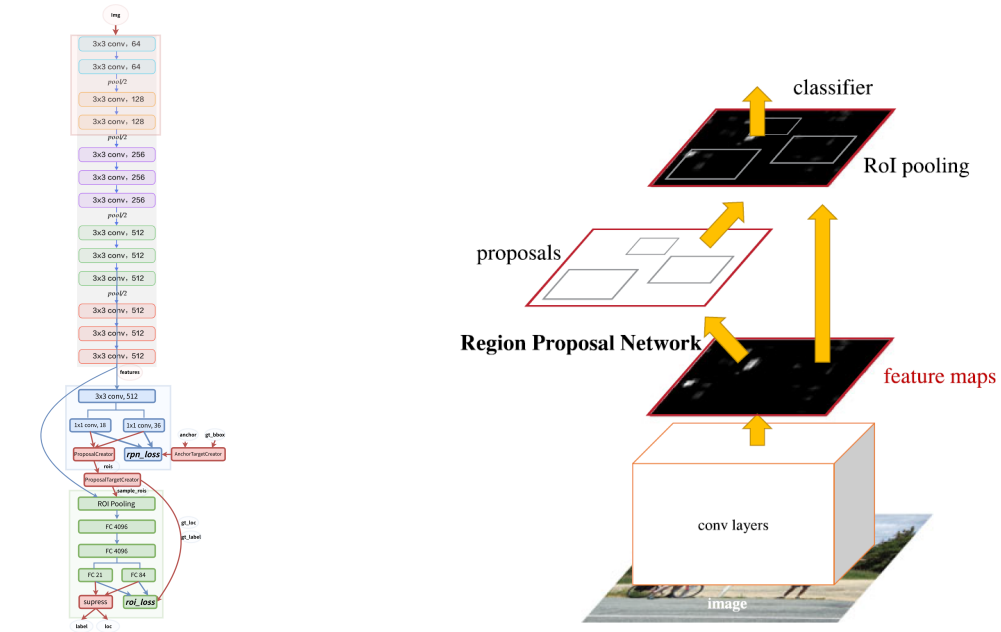
A hyper parameter named mosaic can be set beforehand to set the degree of data augmentation.(the value of mosaic range from 0 to 1, where 0 means no any augmentation)

5 Faster R-CNN

5.1 Principle

In the field of object detection, Faster R-CNN has shown great vitality, which is still the basis of many object detection algorithms to this day. The reason why Faster R-CNN wins is the region proposal algorithm it uses to hypothesize object locations, enabling nearly cost-free region proposals.

Faster R-CNN is composed of two modules (shown in Figure 21(b)). The first module is a deep fully convolutional network that proposes region of interests (also called RoIs), which is also called RPN, and the second module is the RoI head which is in fact the Fast R-CNN detector using the RoIs to recognize objects. By combining the region proposal module and the detection module together into a single network and sharing the feature map, the model can learn better and faster.



(a) Detailed Layers (b) Overview of the Network

Figure 21: Network structure

5.1.1 Preprocessing of data

Before fed into the network, every image needs to be preprocessed to train the network. First, it will be resized, making the longer side of the image is no longer than 1000, the short side of the image is no shorter than 600, and at least one equal sign holds. Correspondingly, the bounding box of the objects should be scaled, too. Finally, subtract the mean of the images pixels, so that the mean of the total image is zero. Finally, the real input into the network are the scaled images with their adapted coordinates of the bounding boxes, the zoom factors and their labels. But when predicting images, only the scaled images and coordinates are needed.

5.1.2 Feature Extractor

The feature extractor module in Faster RCNN is a pretrained model on other dataset, for example, VGG Net, which was used in the author's implementation. Therefore, suppose the size of the input image is 3 $\times$ H $\times$ W, the size of the feature map is C $\times$ (H/16) $\times$ (W/16). These pretrained models can extractor features better than training the network from scratch.

5.1.3 Region Proposal Networks

The most outstanding contribution of Faster RCNN is the proposal of the Region Proposal Network, replacing the Selective Search method when hypothesizing object locations. In this way, the time overhead of generating RoIs is almost reduced to 0 (from 2s to 0.01s).

Before the classification layer and regression layer, the feature map will fist go through a 3 $\times$ 3 convolutional layer to fuse the features better. Then for every points on the feature map, Faster R-CNN predict k (k=9 by default) proposals that may have objects. So the regression layer has 4 k outputs encoding the coordinates of k boxes, and the classification layer outputs $2k$ scores that estimate probability of object or not object for each proposal, where the k proposals are called $anchors$. An anchor is centered at the related point on the feature map with a scale and aspect ratio. In Faster R-CNN, the three scales are 128, 256, 512, and three aspect ratios are 1/2, 1/1, 2/1.

Then, what RPN does is use the AnchorTargetCreator to select 256 anchors form about 20000 anchors to perform classification and regression. The detailed processes are below:

· For each ground truth bounding box, select the anchor with the highest overlap with it as a positive sample.

· For the remaining anchors, select the anchors with an overlap of more than 0.7 with any ground truth bounding box as positive samples, and the number of positive samples does not exceed 128.

· Randomly select anchors whose overlap with any ground truth bounding boxes is less than 0.3 as negative samples. The total number of negative and positive samples is 256.

For each anchor, its ground truth label is either 1 (foreground) or 0 (background) ,and ground truth location t is composed of 4 position parameters (t_x, t_y, t_w, t_h), which is better than direct regression coordinates.

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a}, \tag{19}$$

$$t_w = log(\frac{w}{w_a}), \quad t_h = log(\frac{h}{h_a}), \tag{20}$$

$$t_x^* = \frac{x^* - x_a}{w_a}, \quad t_y^* = \frac{y^* - y_a}{h_a}, \tag{21}$$

$$t_w^* = log(\frac{w^*}{w_a}), \quad t_h^* = log(\frac{h^*}{h_a}), \tag{22}$$

where x, y are the coordinates of the center of the box, and w and h are its width and height. Likewise, x_a, y_a, w_a and h_a are the corresponding values of anchor boxes, and x^*, y^*, w^*, h^* are corresponding values of ground-truth boxes.

While training itself, RPN also provides RoIs to Fast R-CNN as training samples. The process of RPN generating RoIs is as follows:

· For each image, use its feature map to calculate the probability that (H/16) $\times$ (W/16) $\times$ 9 (about 20,000) anchors belong to the foreground, and the corresponding position parameters.

· Select the 12000 anchors with the highest probability.

· Using the regression position parameters, correct the position of the 12,000 anchors to get RoIs.

· Using Non-maximum suppression (NMS), select the 2,000 RoIs with the highest probability.

But when predicting the image, in order to improve the speed, 12,000 and 2,000 anchors become 6,000 and 300 respectively. This process is called ProposalCreator, and this part of the operation does not require backpropagation.

5.1.4 RoI Head

The Region Proposal Network only gives 2,000 candidate boxes, and the RoI Head continues to perform classification and location regression on top of the 2,000 given candidate boxes. Because the provided RoIs corresponds to differnet sizes of areas in the feature map, they can not be fed into the RoI Head directly. Thus, Faster R-CNN first uses the ProposalTargetCreator to select 128 sample RoIs and then uses RoIPooling to map these different sizes of proposals into the same size. After that, reshape them into a one dimensional vector, and use the remaining fully connected layers to perform multi-class classification and location regression.

5.1.5 Loss Function

The loss function of the whole network contains four parts - the regression loss and classification loss of both the Region Proposal Network and the RoI head (or Fast RCNN). Both of the RPN loss and RoI head loss have the form as:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*). \tag{23}$$

Here, L_{cls} is the cross entropy loss, and L_{reg} is the smooth L_1 loss, which is:

$$smooth\ L_1 = \begin{cases} 0.5x^2 & if\ |x| < 1, \\ |x| - 0.5 & if\ otherwise \end{cases} \tag{24}$$

Here, i is the index of an anchor in a batch and p_i is the predicted probability of the anchor being an object. And the ground-truth label p_i^* is 1 if the corresponding anchor is positive and is 0 otherwise. Besides, the loss is normalized by the batch size N_{cls} and the number of anchor locations N_{reg}, and weighted by a balancing parameter λ (default to 10). To compute the loss of RoI head, the anchors above should be replaced by RoIs, and the number of classes should be changed from 2 to 21 (20 classes plus the background).

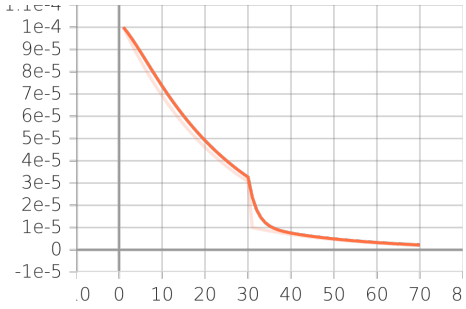
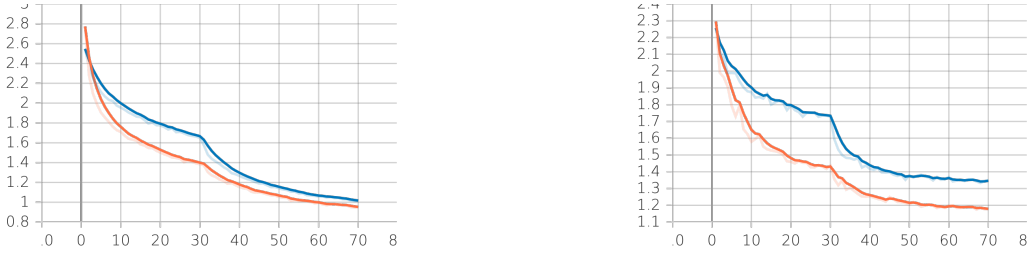


Figure 22: learning rate throughout training



(a) training loss (b) validation loss

Figure 23: Total loss throughout the training

5.2 Experiments

5.2.1 Experiment settings

Considering that the backbone of the Faster R-CNN can be both Resnet50 or VGG, it is natural to think about the detection results based on the two different backbones. SO, we conducted the experiments with these two backbones, but the others hyperparameters are the same. In our experiments, when training the network, we first freeze the backbone and fine-tuning for 30 epochs, and then unfreeze the backbone and keep training for 40 epochs. Other detailed settings are as follows:

Freeze stage:

· Batch size: 4

· Iteration: 2505 iterations when training and 246 iterations when validation.

· Optimizer: Adam.

· Weight decay: 0.0005

· Initial learning rate: 0.0001

· Learning rate decay: $lr = \lambda \times lr$ ($\lambda = 0.96$, updates every one epoch)

Unfreeze stage:

· Batch size: 2

· Iteration: 2505 iterations when training and 2476 iterations when validation.

· Optimizer: Adam.

· Weight decay: 0.0005

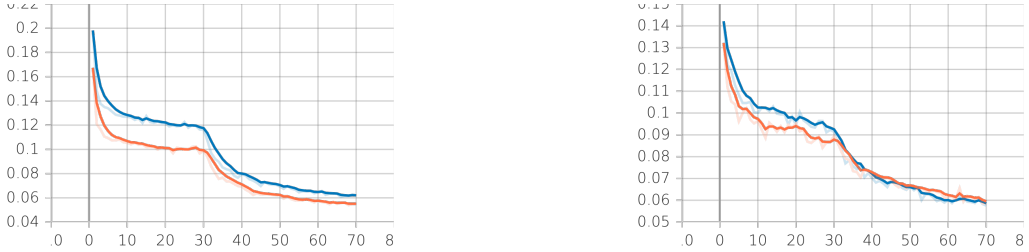
· Initial learning rate: 0.00001

· Learning rate decay: $lr = \lambda \times lr$ ($\lambda = 0.96$, updates every one epoch)

Changes in learning rate throughout the training procecss can be seen in Figure 22. The dataset for training we use are the training set and validation set in VOC2007, and the validation set is the testing set in VOC2007. For testing, we use the validation set in VOC2012.

5.2.2 Loss

During the training process, the losses on the training set and the validation set are shown in Figure 23. The blue curves represent the losses based on VGG, and the orange curves represent the losses based on the Resnet50 (the following is also the case). As we can see, the losses of the Resnet50-based network are higher than those of VGG-based network in general, which may due to the more parameters VGG network has.

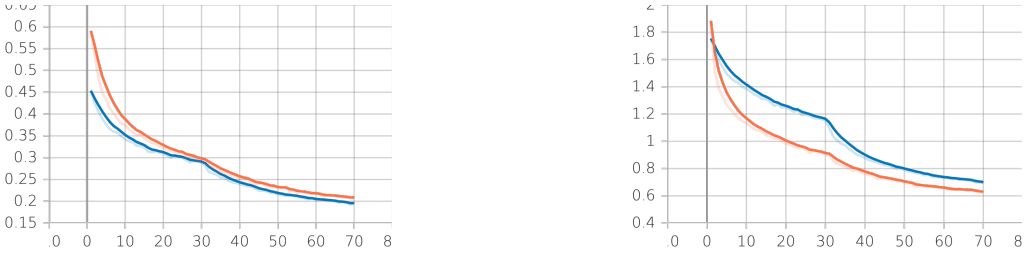


(a) RPN classification loss

(b) RPN location loss

Figure 24: RPN loss

Now, let me dive into more detailed information of the loss. As we can see in Figure 24 and Figure 25, the main component of the loss from the RoI module, especially the RoI location loss. When the learning rate is relatively bigger (in first 30 epochs), the gap between the two network is a little larger, and when the learning goes smaller (in the last 40 epochs), the VGG-based network catches up slowly, even better.

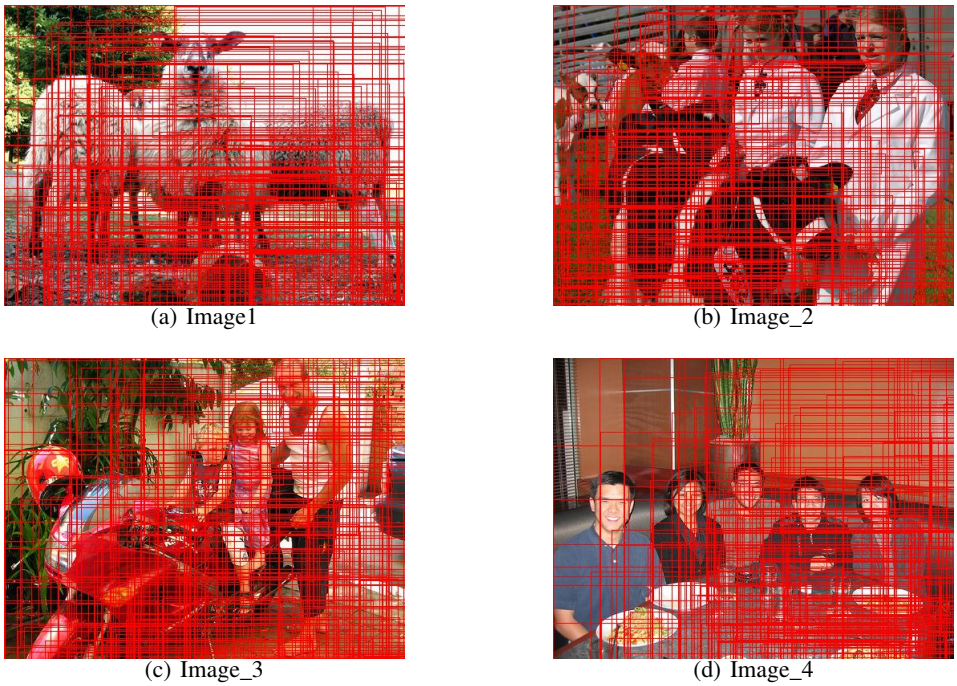


(a) RoI classification loss (b) RoI location loss

Figure 25: RoI loss

5.2.3 Visualization

We select four images in VOC2007 dataset, and visualize the 300 generated proposal boxes, and the results are shown in Figure 26. As we can see, areas with objects have more proposal boxes than other areas in the picture, and the proposal boxes almost cover all the objects in the image.



(a) Image1 (b) Image_2

(c) Image_3 (d) Image_4

Figure 26: The proposal boxes from the first stage

5.2.4 Evaluation metrics

The mAP@.5 throughout training process has been shown in Figure 27, and the mIOU and acc throughout training have been shown in Figure 28(a) and Figure 28(b). If we compare some evaluation metrics between the two networks, we can easily find that VGG-based network is greater than Resnet50-based network when training 70 epochs, even through the loss of VGG-based network is bigger than another.

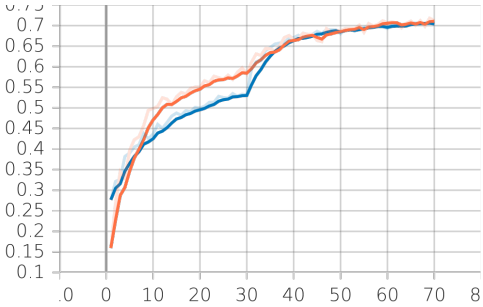
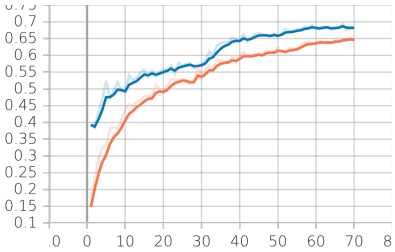
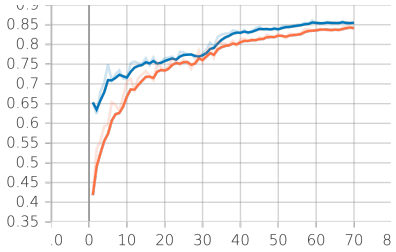


Figure 27: mAP@.5 throughout training



(a) mIoU



(b) acc

Figure 28: mIoU and acc

6 Comparison of Faster RCNN and YOLOv3

The current object detection algorithms can be roughly divided into two categories, namely two-stage and one-stage.There are also multi-stage series of algorithms before, but due to the gap in speed and accuracy, they have been rarely used.

The two-stage detection algorithm divides the detection problem into two stages, first generating candidate regions, and then classifying candidate regions after location refinement. The two-stage detection algorithm has a relatively low recognition error rate and a low missed recognition rate, but the speed is slow and cannot meet real-time detection scenarios, such as video object detection.

On the other side, The one-stage detection algorithm does not need to generate a candidate region stage, but directly generates the category probability and position coordinate value of the object. After a single detection, the final detection result can be directly obtained, so it has a faster detection speed.However, the general recognition accuracy is usually worse than the two-stage algorithm.

Faster-RCNN and YOLO V3 are two representative models in the two types of algorithms.In the experiments above, we finally get several metrics of these 2 models after training, validating and testing on VOC dataset.The metrics are shown below(Although we trained YOLOv3 for 5 more epochs in the experiment, the metrics of YOLOv3 below are also got from models **trained with 70 epochs**) As we can see, the YOLOv3 has obviously higher mAP and mIoU than

Models(70 epochs)	mAP(@.5)	MIoU	Accuracy
Faster RCNN(based on ResNet50)	0.649	0.609	0.806
Faster RCNN(based on VGG16)	0.644	0.643	0.821
YOLOv3	0.896	0.712	0.795

Table 3: Comparison between YOLOv3 and Faster RCNN

Faster RCNN, and they have almost the same Accuracy. In terms of the time consumptionthough we didn't record the

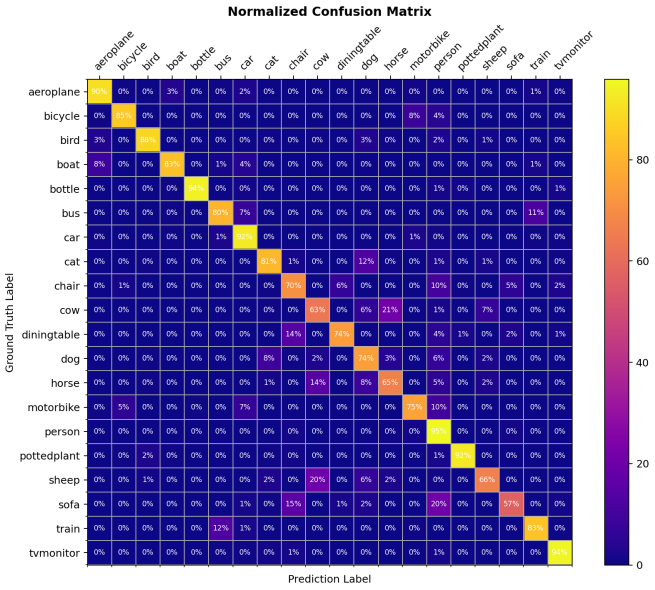


Figure 29: confusion matrix

precise information, we can judge their speed roughly from the fact that YOLOv3 only took 3-4 hours to train while Faster RCNN at least took a whole night.Therefore, the experimental results prove the theory mentioned above.

To compare the inference ability of two models, we let then make inferences about and visualize 3 same images(in the first row of figure30).We finally got 12 images and we displayed them in figure 30:

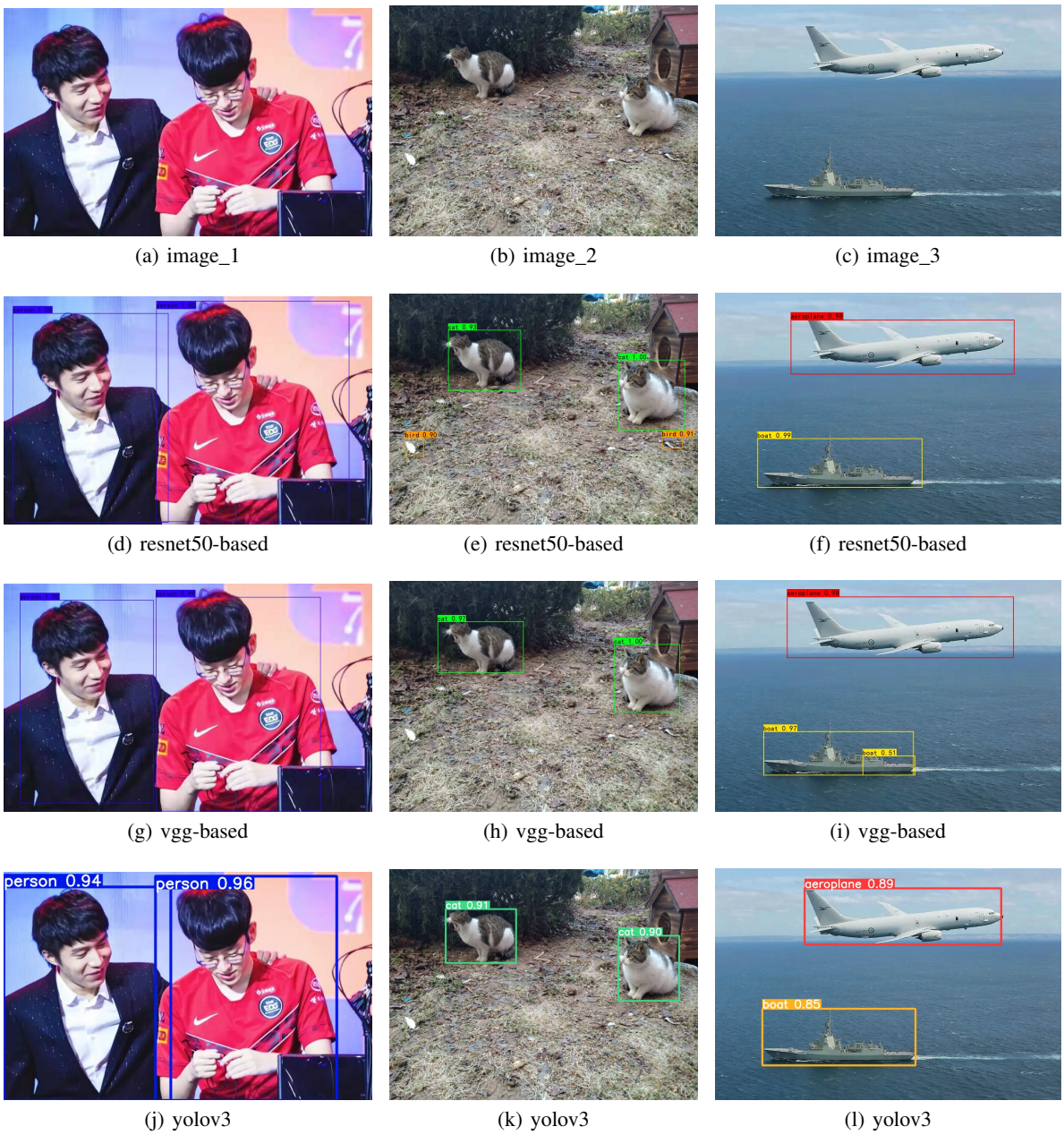
As we all know, Compared with V1 and V2, the YOLO V3 algorithm has indeed made great improvements, especially in small object detection and overlapping object detection. It has made great breakthroughs, and its generalization ability may be strong.In the Figure30 we can simply find that Faster RCNN mistakes the small litter as birds in the second image and draw an extra box when identifying the **ship**(which has a little difference form **boat** in VOC dataset) in Figure 3. This proves that YOLOv3 indeed performs better in small object detection and possesses greater generalization ability.

Nevertheless,if we take a closer look, it's easy to find that the scores of right predicted objects are all higher than those predicted by Yolov3. Although the score difference is not significant, it does show the advantages of one-stage detection.

7 Conclusion For Faster RCNN and YOLOv3

In the project, we trained and validated Faster RCNN and YOLOv3 on VOC2007 dataset and tested on VOC2012 dataset. We used tensorboard to plot metrics such as mAP,mIoU and Accuracy, which can be viewed specifically in the above parts. Besides, we optionally plotted confusion matrix or P-R curve to visually demonstrate those metrics.Through the train-validation-test process, we find that YOLOv3 has a faster speed and have higher mAP and mIoU than Faster RCNN.But that didn't determine their inference ability.

Furthermore, In order to compare these 2 models(YOLOv3 and Faster RCNN), we used them to make inferences about 3 same pictures and visualized the result in the last part. Through the comparison,We prove that while YOLOv3 do better in small object detection and possesses more generalization ability, Faster RCNN has higher scores when doing inference tasks.



(a) image_1 (b) image_2 (c) image_3

(d) resnet50-based (e) resnet50-based (f) resnet50-based

(g) vgg-based (h) vgg-based (i) vgg-based

(j) yolov3 (k) yolov3 (l) yolov3

Figure 30: Detection results by Faster RCNN and Yolov3

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[2] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.

[3] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *CoRR*, vol. abs/1710.09412, 2017.

[4] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," *CoRR*, vol. abs/1905.04899, 2019.

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[6] R. Girshick, "Fast r-cnn," *arXiv e-prints*, 2015.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[8] P. Purkait, C. Zhao, and C. Zach, "Spp-net: Deep absolute pose regression with synthetic views," 2017.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *IEEE*, 2016.

[10] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv e-prints*, 2018.

[11] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Large scale learning of general visual representations for transfer," *CoRR*, vol. abs/1912.11370, 2019.

[12] "The pascal visual object classes homepage." `http://host.robots.ox.ac.uk/pascal/VOC/`.

[13] K. He, R. B. Girshick, and P. Dollár, "Rethinking imagenet pre-training," *CoRR*, vol. abs/1811.08883, 2018.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[15] F. Ramzan, M. U. Khan, A. Rehmat, S. Iqbal, T. Saba, A. Rehman, and Z. Mehmood, "A deep learning approach for automated diagnosis and multi-class classification of alzheimers disease stages using resting-state fmri and residual neural networks," *Journal of Medical Systems*, vol. 44, 12 2019.

[16] A. Chakure, "All you need to know about yolo v3 (you only look once)." `https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m`.

[17] "Principle analysis of yolov3 principle." `https://blog.600mb.com/a?ID=00950-9031089d-f698-40e1-be34-66b0a9be6457`.

[18] W. zhe, *Research on detection, recognition and tracking of ships in waterway under dynamic background based on deep learning*. PhD thesis, Sanxia University.

[19] "Miou calculation.computation of miou for multiple-class based semantic image segmentation." `https://medium.com/@cyborg.team.nitr/miou-calculation-4875f918f4cb`.

[20] T. Liu and L. I. Dongsheng, "Detection method for sweet cherry fruits based on yolov4 in the natural environment," *Asian Agricultural Research*, vol. 14, no. 1, p. 7, 2022.