Homework 2

Visual Recognition using Deep Learning
Object Detection

Department : IMM, Yi-Cheng Hung, ID : 310653005

September 18, 2022

Github

Github : https://github.com/YCHung1998/Object-Detection.git

Introduction

This task is to identify the numbers in the SVHN data set and select the corresponding coordinate range. The goal is to achieve a certain level of average accuracy and recognition rate. In terms of model considerations and complexity, it is necessary to select and analyze the method more carefully. For the measurement problem, RCNN was used at the beginning. As the years are updated, more advanced methods such as faster-RCNN, YOLO (You Only Look Once) accelerate the reading rate and improve the new architecture model of mAP(mean of average precision). In the following, We will introduce our method.

Figure 1: SVHN example

Data
Before sorting out a certain piece of data, there is a problem with the correspondence of bbox **(12668.png)**. Some adjustments have been made to this part of the data. After adjustment, everything went well.

Thinking
Although the score threshold this time seems to be very low. But in the field of real-time object detection, it is not easy to pass this threshold. Since I am not familiar with the method of object detection, I will start with the faster-rcnn explained in the class.

But just forecasting is no longer the main concern. We also pay attention to speed. In terms of calculation, the training time of the rcnn family is much longer than that of the yolo family. Not only can't adjust the MBS, but also in the case of huge data volume. In contrast, the calculation time and training time will be several times. Following the Faster-RCNN example to coding by self, it takes lots of time and not have a good performance. After that, start take the **torchvision.models.detection.faster_rcnn**[1] to give some trial, but the situation is not optimistic. This is not the main method. So starting with the model of the yolo family, the most troublesome thing about this is to use the program code opened on github to crack. (I'm not very good at using it, I just started to step into this field)

YOLO structure

The YOLO network consists of three main pieces.

1. Backbone - A convolutional neural network that aggregates and forms image features at different granularities.
2. Neck - A series of layers to mix and combine image features to pass them forward to prediction.
3. Head - Consumes features from the neck and takes box and class prediction steps.

This paragraph introduces the Yolo architecture in different periods and describes the latest methods used. It may cover yolov1∼5. It may be too long to write, so focus on the main points.

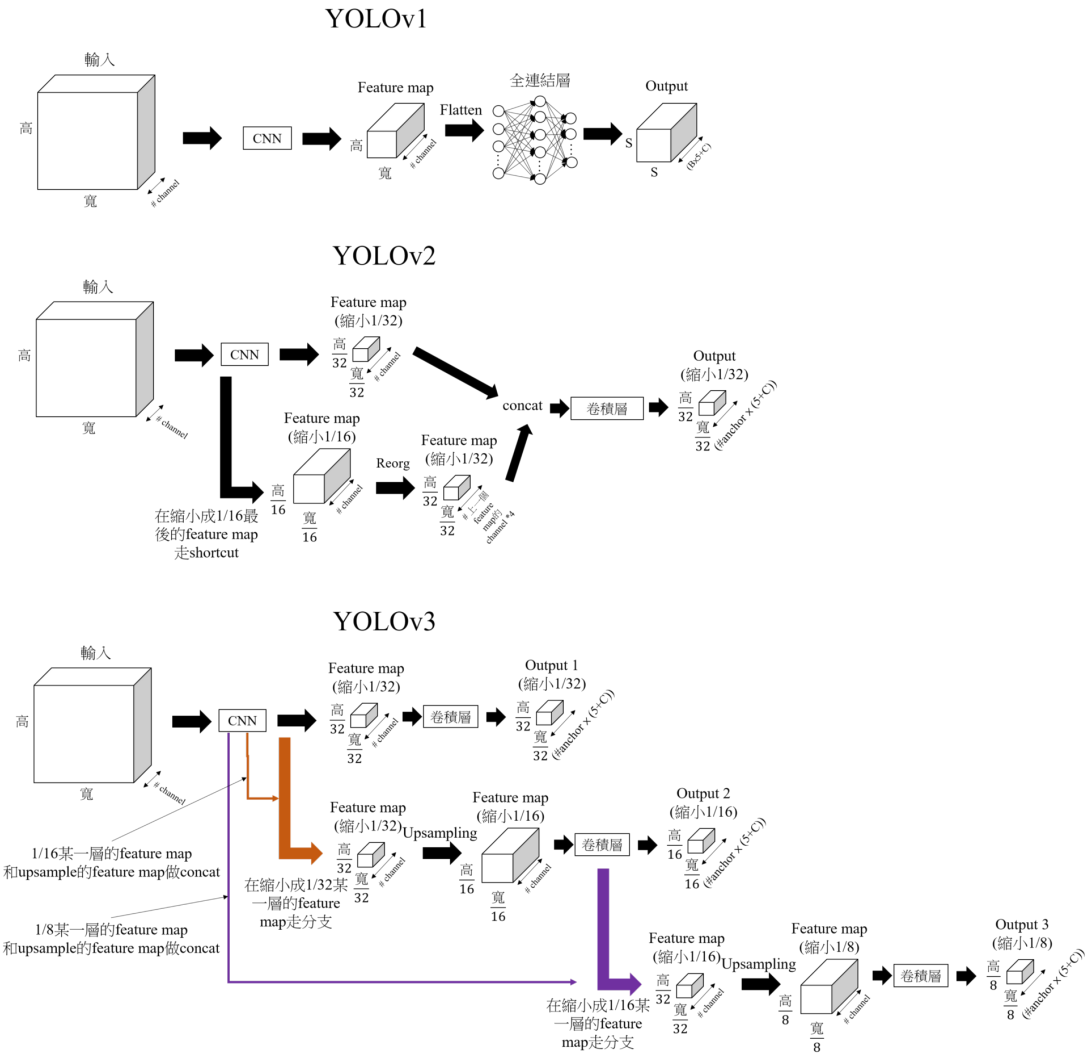


Figure 2: Base on yolov1, yolov2 create more different size feature map, replace the full connected layer into convolution layer and output the smaller size but more channels of result. In yolov3, extract more feature maps and output the three terms and the bottom concate together . Specially, both yolov2 and yolov3 introduce the anchor design of Faster RCNN, so the last layer is the output of the convolutional layer. This is why we need to start from Faster-rcnn.(figure source ref.6)

- Adaptive anchor frame calculation
  In the Yolo algorithm, for different data sets, there will be anchor boxes with initial length and width settings. In network training, the network outputs the prediction frame based on the initial anchor frame, and then compares it with the groundtruth of the real frame, calculates the gap between the two, and then reverses the update to iterate the network parameters.

- Mosaic data enhancement
  The input of Yolov5 uses the same Mosaic data enhancement method as Yolov4. According to the author, the stitching uses random scaling, random cropping, and random arrangement. The detection effect of small targets is still very good, which is very suitable for this problem.(Visaulize the view under the figure)

Figure 3: Mosaic

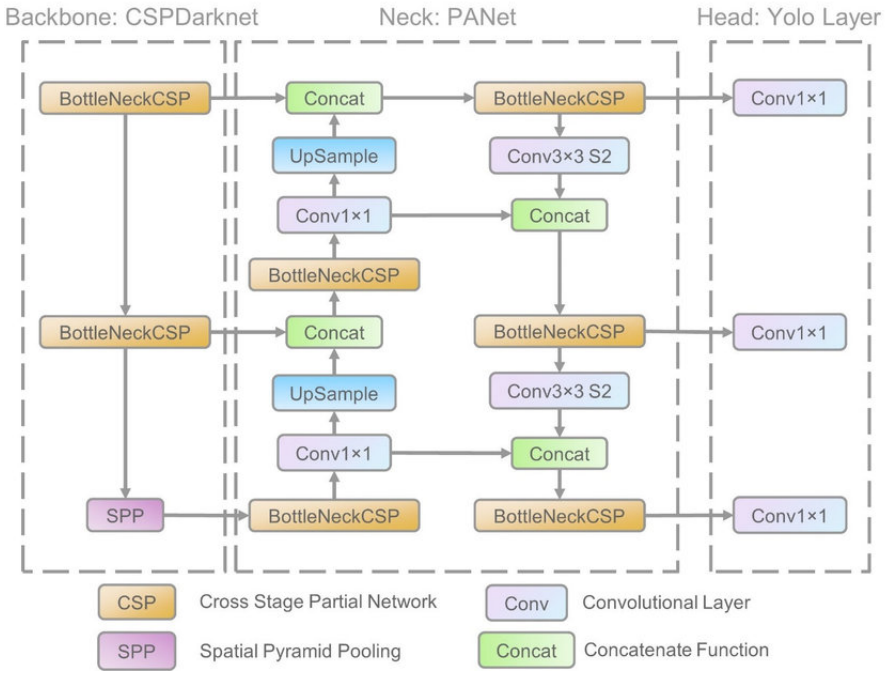


Figure 4: YOLOV5 architecture

Model select

I take yolov5m (YOLO V5 medium) and pretrain weight to train our task. I chose a medium size model because the amount of parameters is not too much, the recognition rate per second is also very fast, and because the data and photos are very small this time, 640 should be sufficient when adjusting the pixel size. In the result, this model performance is satisfactory.(Only 20 Epoch training beat the baseline)

Model	size	AP^{val}	AP^{test}	AP_{50}	$Speed_{V100}$	FPS_{V100}	params	GFLOPS
YOLOv5s	640	36.8	36.8	55.6	**2.2ms**	**455**	7.3M	17.0
YOLOv5m	640	44.5	44.5	63.1	2.9ms	345	21.4M	51.3
YOLOv5l	640	48.1	48.1	66.4	3.8ms	264	47.0M	115.4
YOLOv5x	640	**50.1**	**50.1**	**68.7**	6.0ms	167	87.7M	218.8
YOLOv5x + TTA	832	**51.9**	**51.9**	**69.6**	24.9ms	40	87.7M	1005.3

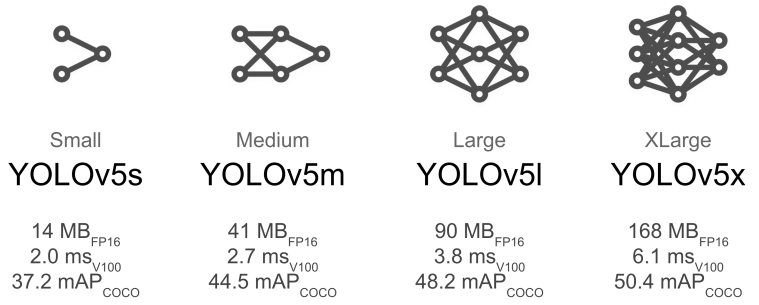


Figure 5: We can download pre-trained weight here

Training
The following indicator and hyper-parameters all we can modify.

1. python train.py −−batch 16 −−cfg yolov5m.yaml −−weights ” −−data SVHN.yaml −−img 640 −−epochs 20

2. param

lr0: 0.01	initial learning rate (SGD=10^{-2})
lrf: 0.1	final OneCycleLR learning rate (lr0 $*$ lrf)
momentum: 0.937	SGD momentum
weight_decay: 0.0005	optimizer weight decay 5×10^{-4}
warmup_epochs: 3.0	warmup epochs (fractions ok)
warmup_momentum: 0.8	warmup initial momentum
warmup_bias_lr: 0.1	warmup initial bias lr
box: 0.05	box loss gain
cls: 0.5	cls loss gain
cls_pw: 1.0	cls BCELoss positive_weight
obj: 1.0	obj loss gain (scale with pixels)
obj_pw: 1.0	obj BCELoss positive_weight
iou_t: 0.20	IoU training threshold
anchor_t: 4.0	anchor-multiple threshold
anchors: 3	anchors per output layer (0 to ignore) ll
fl_gamma: 0.0	focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015	image HSV−Hue augmentation (fraction)
hsv_s: 0.7	image HSV−Saturation augmentation (fraction)
hsv_v: 0.4	image HSV−Value augmentation (fraction)
degrees: 0.0	image rotation (+/- deg)
translate: 0.1	image translation (+/- fraction)
scale: 0.5	image scale (+/- gain)
shear: 0.0	image shear (+/- deg)
perspective: 0.0	image perspective (+/- fraction), range 0∼0.001
flipud: 0.0	image flip up−down (prob.)
fliplr: 0.5	image flip left−right (prob.)
mosaic: 1.0	image mosaic (prob.)
mixup: 0.0	image mixup (prob.)
copy_paste: 0.0	segment copy−paste (probability)

loss function

The loss function in yolov1. The first two terms are checking the Bounding box Location (x, y) when there is object and mainly adjusted the Bounding box size width and height when there is object. Next two terms show that the confidence of the object in bounding box or not in ,respectivly. In the last term to classify this objection belongs to something. And given some weight to balance every term. The following loss or cost function is the ancient method (yolov1 2015).

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

But the above loss function just for the oldest method. Although yolov1 adjust the bbox loss by root, it might still move too large. In yolov2 show the new way to correct this problem.

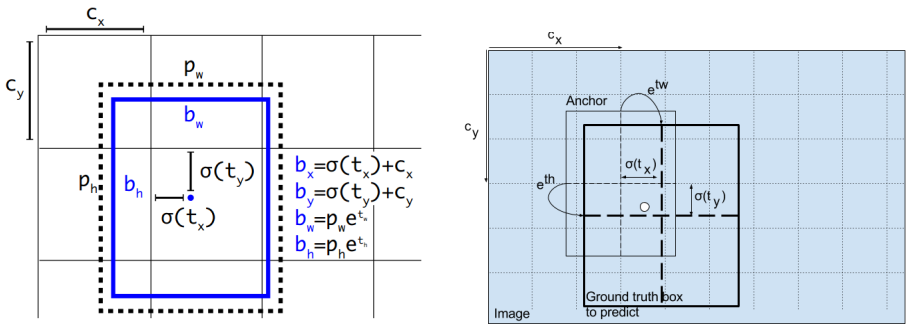


Figure 6: c_x, c_y is the number of the cell in between horizontal and vertical distance the box's center and the top left corner in uni of anchor width and height respectively. Then the second row will be renewed.(If you want to learn more, please open the yolov2 paper)

In YOLOv5 loss/cost function, there are more diversified combinations that we can apply. Total loss also separate by three parts. l_{bbx}, l_{obj} and l_{cls} for **bounding box regression** default is CIoU.(also have DIoU, GIoU can choose, but we use default setting.)
A good bounding box regressor contains three elements:

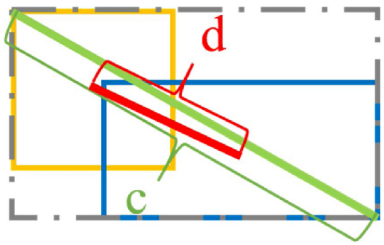
1. Overlapping area
2. Central point distance
3. Aspect ratio

$\mathcal{L}_{C\,\mathrm{IoU}}$: CIoU Loss defined as

$$v = \frac{4}{\pi^2}\left(\tan^{-1}(\frac{w^{gt}}{h^{gt}}) - \tan^{-1}(\frac{w}{h})\right)^2, \qquad \alpha = \frac{v}{(1-IoU)+v}$$

$$\mathcal{L}_{C\,\mathrm{IoU}} = 1 - IoU + \frac{\rho^2\left(\mathbf{b}, \mathbf{b}^{gt}\right)}{c^2} + \alpha v$$

ρ : Euclidean distance 2−norm
c : the maximum diagonal distance of the prediction and Ground Truth.
$c = \max_{v_p(i), v_T(j)} D(v_p(i), v_T(j))$
$d = \dfrac{\rho^2\left(\mathbf{b}, \mathbf{b}^{gt}\right)}{c^2}$



Objection loss :

BCEWithLogitsLoss = Sigmoid + BCELoss

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -w_n\left[y_n \cdot \log\sigma\left(x_n\right) + \left(1 - y_n\right) \cdot \log\left(1 - \sigma\left(x_n\right)\right)\right]$$

Classification loss:

BCELoss

Focal loss, QFocal loss also the loss method in the yolov5 package we can choose. We still maintain original setting. Inthe future work. We'll try more method to robust the prediction model and implement in some kind of field(Maybe Medical Image Phase 1 prediction).

Result
Under the following pictures show the different indicator to evaluate the prediction performance in the classification.

- TP : Truth positive
- FP : False positive
- FN : False negative

1. Precision : $\frac{TP}{TP+FP}$.
2. Recall : $\frac{TP}{TP+FN}$.
3. F1−score : harmonic mean of Precision and Recall.
4. Confidence : the insurance of the objection is that number(prob.)

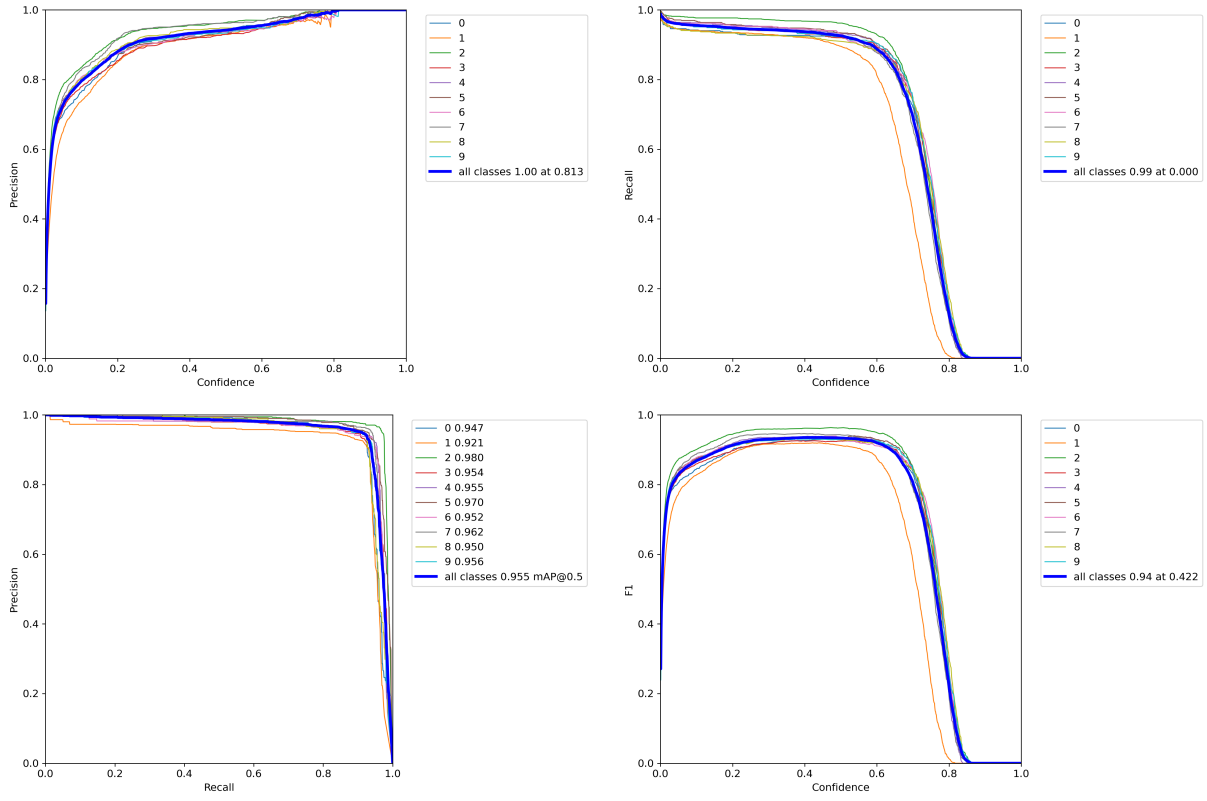


Figure 7: In top left picture, we expected that the curve is closer to the upper left corner. The bottom left one, we hope the curve get more close to upper right corner.

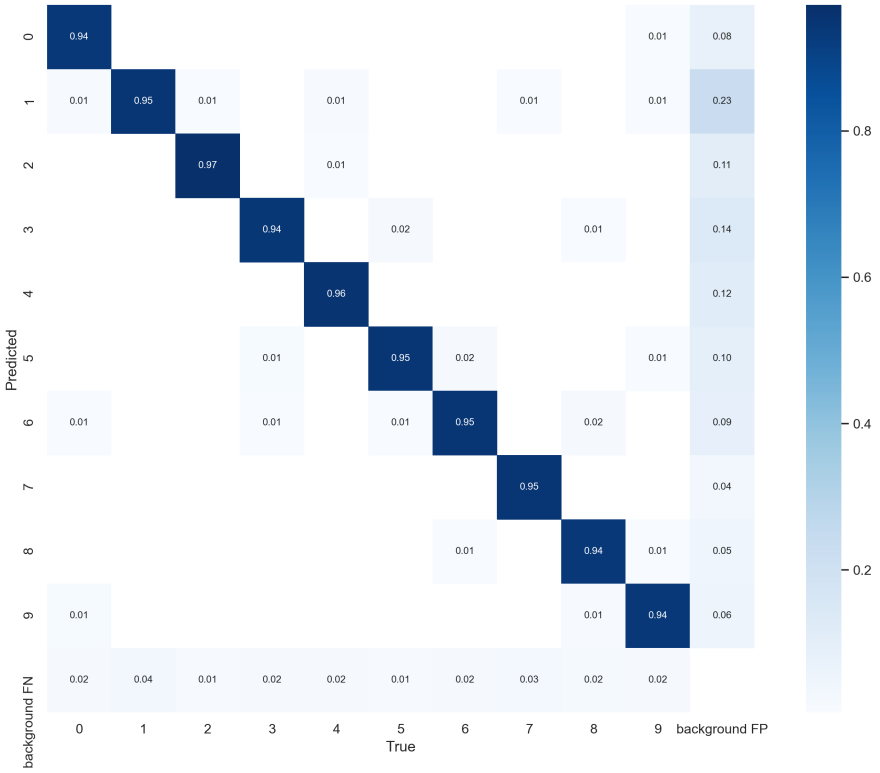


Figure 8: Confusion matrix

we can clearly see that this confusion matrix is diagonally dominant, in the classify way performance very well, only in a small percentage of validation sample predict incorrect.

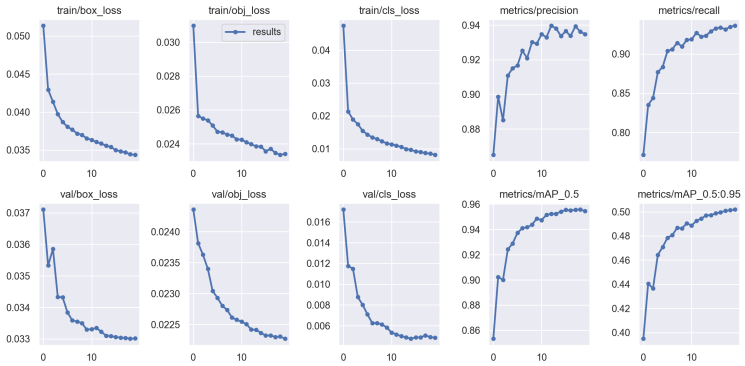


Figure 9: pre-trained weight trained in 20 epochs with MBS=16

Colab Timing

In the following picture shows that the model prediction per image time(Take the average of the predicted time of 100 testing images). I used **LoadImage** function (written in yolov/-datasets) to load image, and this function is also used **cv2.imread**.

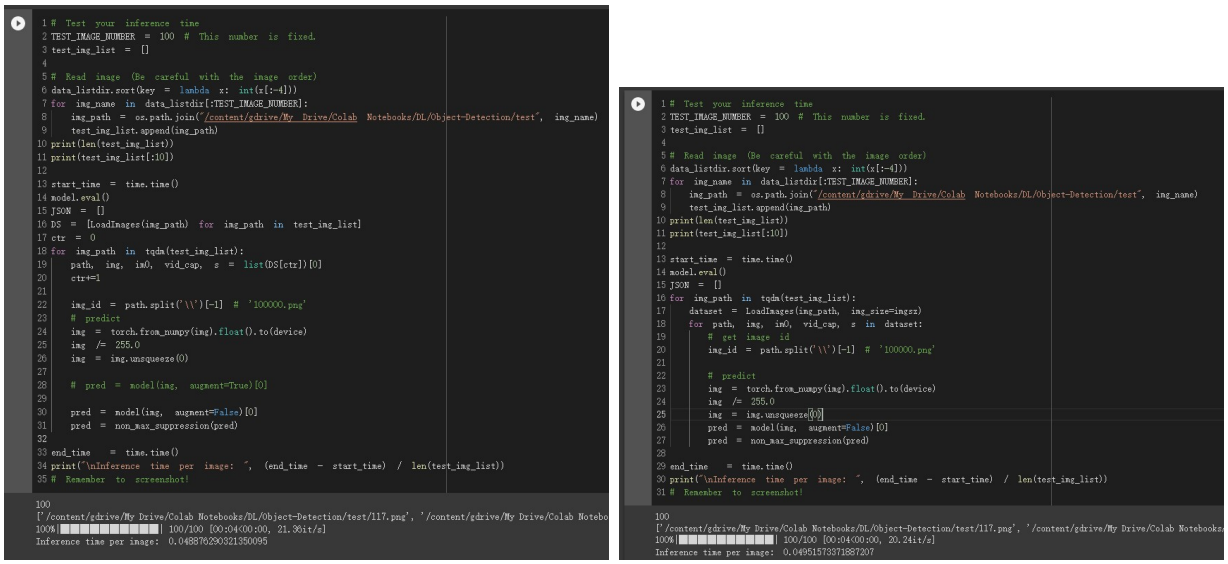


Figure 10: The left hand side figure show that only calculated the model prediction time : 0.04887, the other is incorporate the process of reading photos into the calculation time : 0.04951.

	per image(sec)
Timing(excluded loading)	0.04887
Timing(Including loading)	0.04951

10	0.415967	answer.zip	11/22/2021 16:49:21	Finished		+
11	0.372227	answer.zip	11/23/2021 05:48:39	Finished		+
12	0.410114	answer.zip	11/24/2021 10:35:31	Finished		+
13	0.398339	answer13.zip	11/24/2021 18:49:01	Finished		+
14	0.410297	answer012.zip	11/25/2021 06:24:55	Finished		+
15	0.404188	answer012f.zip	11/25/2021 06:50:46	Finished		+
16	0.410114	answercheckf.zip	11/25/2021 06:58:33	Finished	✓	+
17	0.415967	answercheckt.zip	11/25/2021 07:12:39	Finished		+

Figure 11: Result : Although No.17 mAP is higher than No.16, we still considered the speed of the model prediction, No.17 has augmentation in prediction, No.16 without aug. In the speed of this two model, No.16 is faster than No.17 about 0.1 sec per-image

Visualize

- Under the visualization

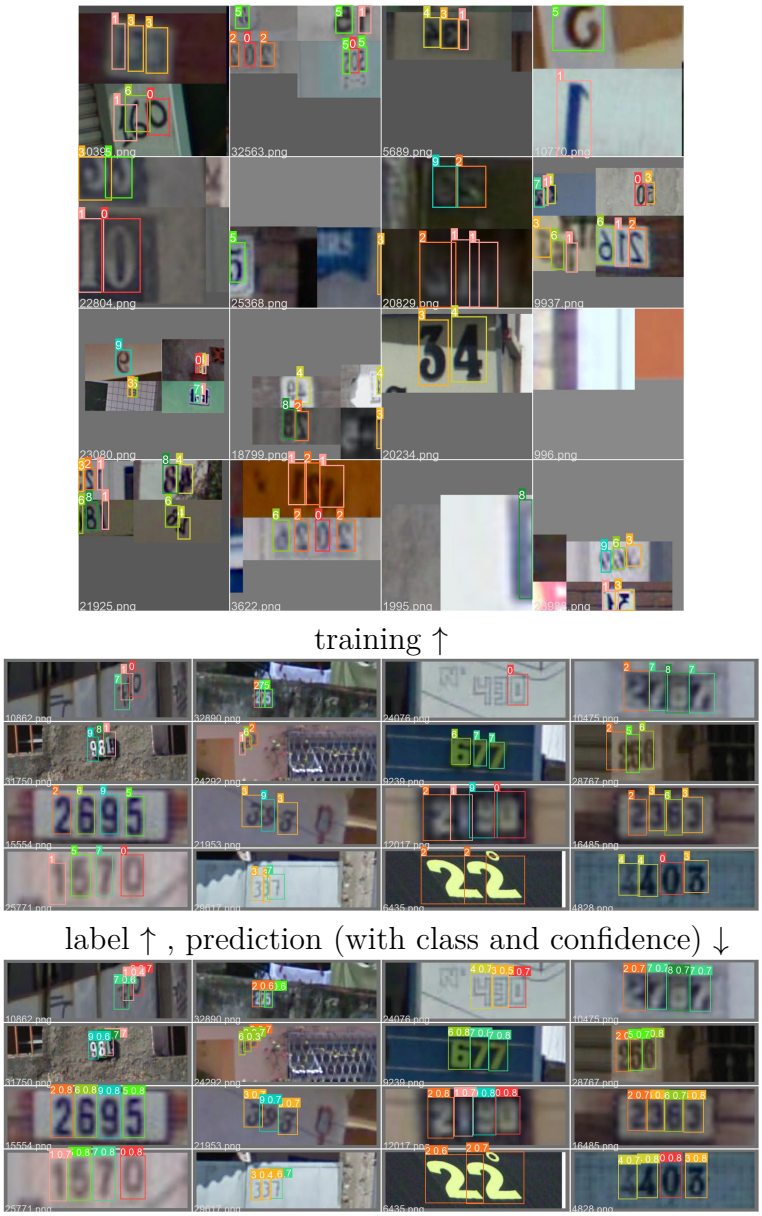


Figure 12: Visualize of training and validation with label, bbx and prediction bbx with confidence score

Conclusion

It took me a lot of time to try and error in this task. Grafting open source materials is not an easy task to me. Because of the unfamiliarity with the format, the process of using it is very difficult. Nevertheless, it is gradually succeeded under discussion among peers. In this report, there are fewer attempts, mainly focusing on the use of the yolov family. With this experience, I will save a lot of time in the future.

Reference

1. Faster-RCNN example
2. Faster r-cnn: Towards real-time object detection with region proposal networks, arXiv:1504.08083v2 [cs.CV] 27 Sep 2015. link
3. You Only Look Once: Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, paper. Code : link.
4. YOLO9000: Better, Faster, Stronger Joseph Redmon, Ali Farhadi, paper. Code : link.
5. YOLOv3: An Incremental Improvement, paper. Code : link.
6. 深度學習-物件偵測 YOLOv1、YOLOv2 和 YOLOv3 cfg 檔解讀 (一), Tommy Huang. yolov1-v3
7. YOLOv5 学习总结