# VRDL HW2 Report

310551055 鄭伯俞

- Github && reference
  - Github Repo
  - Reference(code)
  - YOLOv5 Explaination
  - YOLOv4 Paper
  - Comparison Between YOLOv4 and YOLOv5

Instructions for reproducing my submission are provided on Github.

- Brief introduction

In HW2, I use the Yolov5 implementation on the given digit detection task. YOLOv5 is implemented in pytorch, and has very similar architecture to YOLOv4. By adjusting hyper-parameters, I achieved **0.41477 mAP@0.5~0.95iou** and **0.083s per image** timing on Colab.

The evaluation result is shown below:

Please **screenshot** this cell, including the code and the output (your inference time), and put it into your report.

```
# Test your inference time
TEST_IMAGE_NUMBER = 100 # This number is fixed.
test_img_list = []

# Read image (Be careful with the image order)
data_listdir.sort(key = lambda x: int(x[:-4]))
for img_name in data_listdir[:TEST_IMAGE_NUMBER]:
  img_path = os.path.join("/content/NYCU_VRDL_HW2/datasets/Digits/test/images",img_name)
  img = cv2.imread(img_path)
  test_img_list.append(img)

start_time = time.time()
for img in tqdm(test_img_list):
    # your model prediction
    pred = model(img)
end_time  = time.time()
print("\nInference time per image: ", (end_time - start_time) / len(test_img_list))

# Remember to screenshot!
```

```
100%|██████████| 100/100 [00:08<00:00, 11.95it/s]
Inference time per image:  0.08378937244415283
```

# Methodology

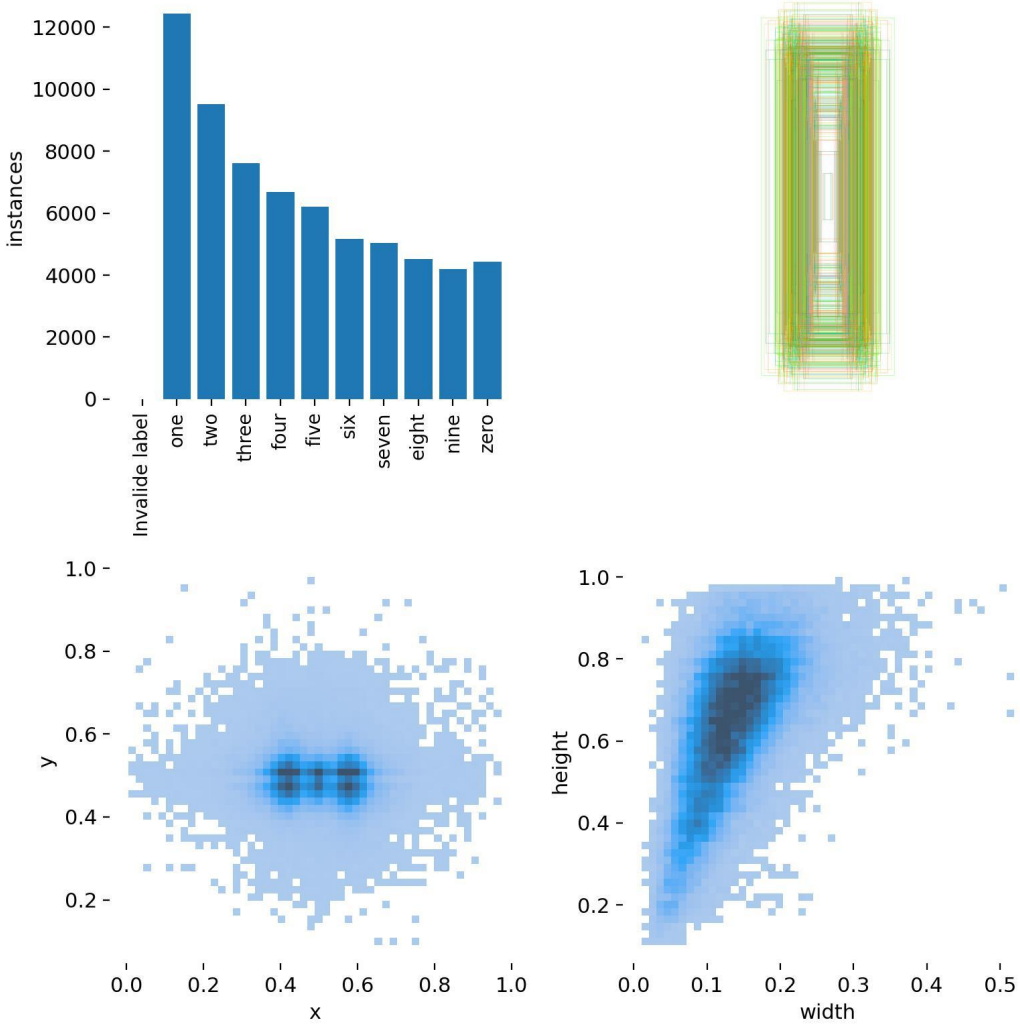
## Data Preprocessing

In the preprocessing step, I first generate an annotation.txt from the .mat file using the online version of matlab, then I use python script to convert the dataset into COCO format, and finally, **I randomly sampled 10% of training image as validation set**.

The preprocessing steps (include download and unzip ) can be performed by **simply typing "make getdataset" in my repo**.

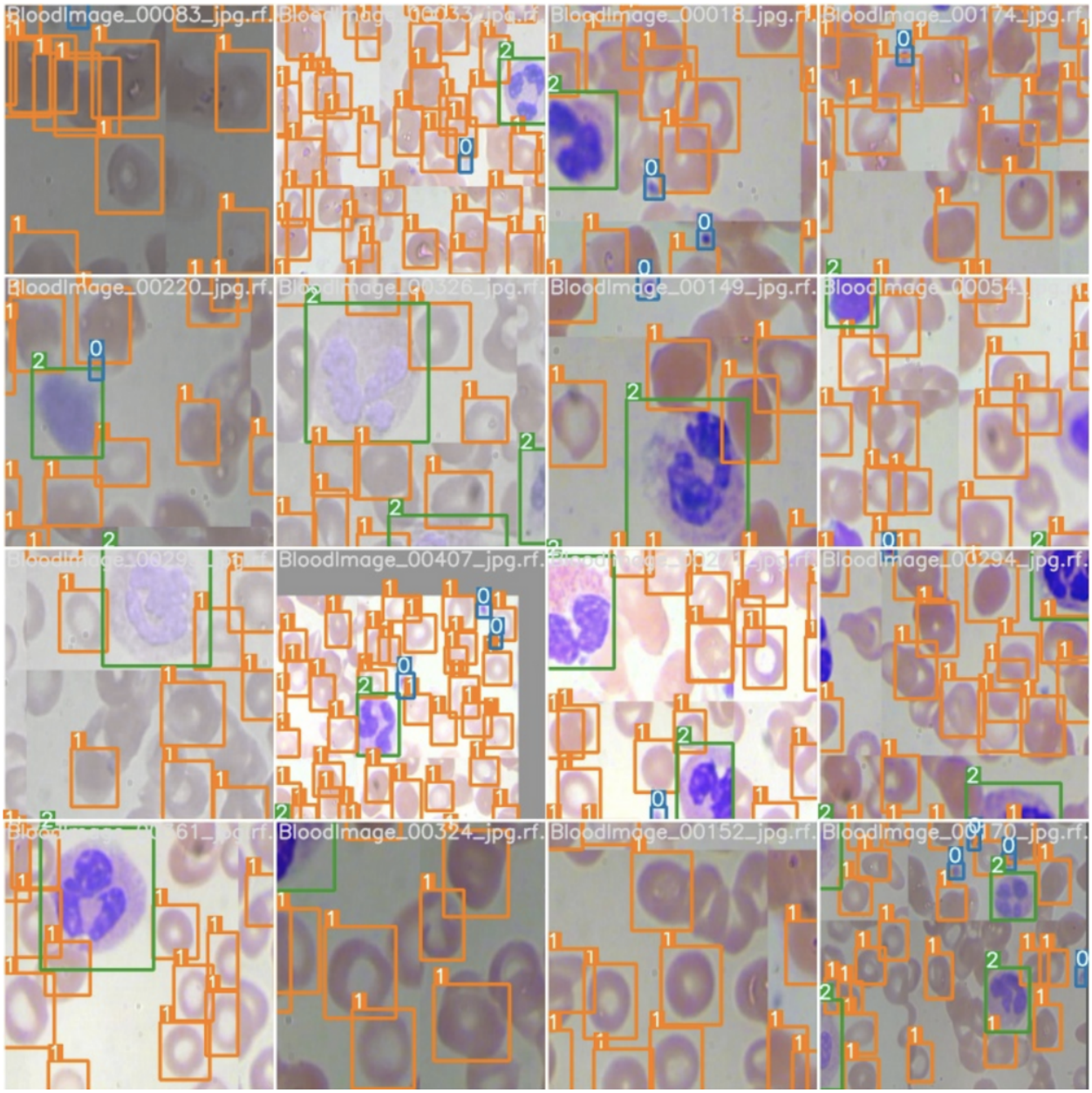
For data augmentation, **I turn off the default fliplr and flipud options** because digits doesn't survive flipping.

For Color augmentation such as hsv and gamma, I use the default value for fine tuning VOC dataset.

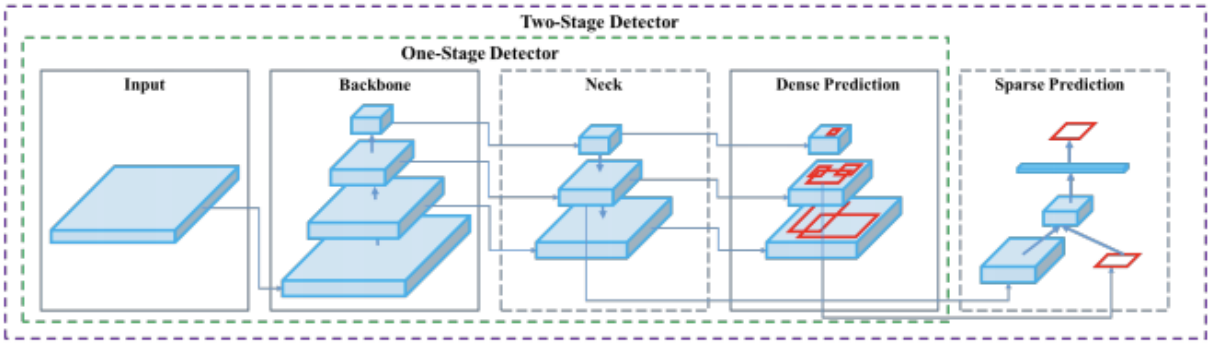


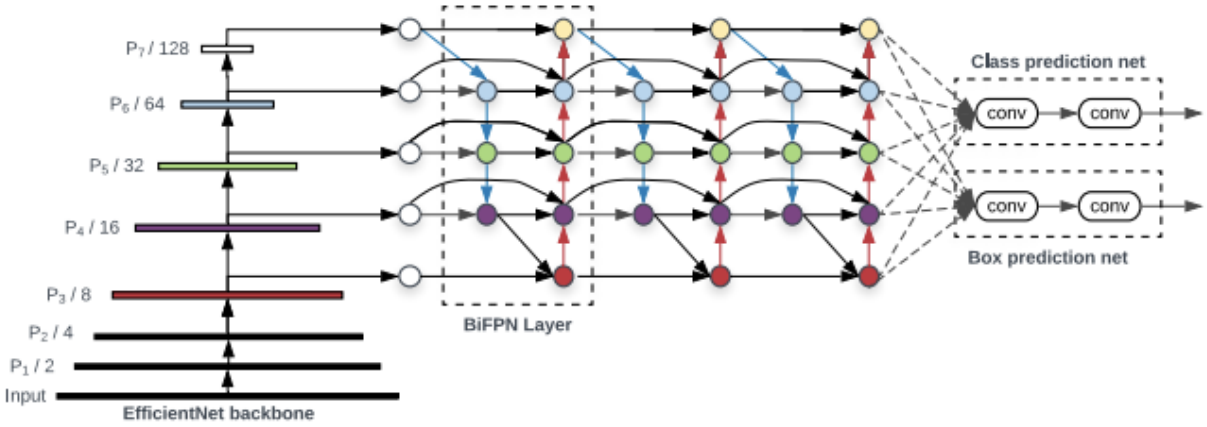
YOLOv5 implementation automatically detect distribution of bounding boxes and chooses proper anchar boxes dimension automatically. The distribution is shown in the previous page.

Finally, YOLOv5 automatically does mosaic augmentation (the method is originated from YOLOv3), as shown below



## Model Architecture




The model architecture is described in the [YOLOv4 paper] , they incorporate multiple existing ideas like Weighted-Residual-Connections(WRC),Cross-Stage-Partial-connections (CSP), Cross mini-BatchNormalization (CmBN), Self-adversarial-training (SAT)and Mish-activation. They also proposed new features such as WRC, CSP,CmBN, SAT, Mish activation, Mosaic data augmentation,CmBN, DropBlock regularization, and CIoU loss. The combination of all these features gives amazing SOTA results.

## Hyperparameters

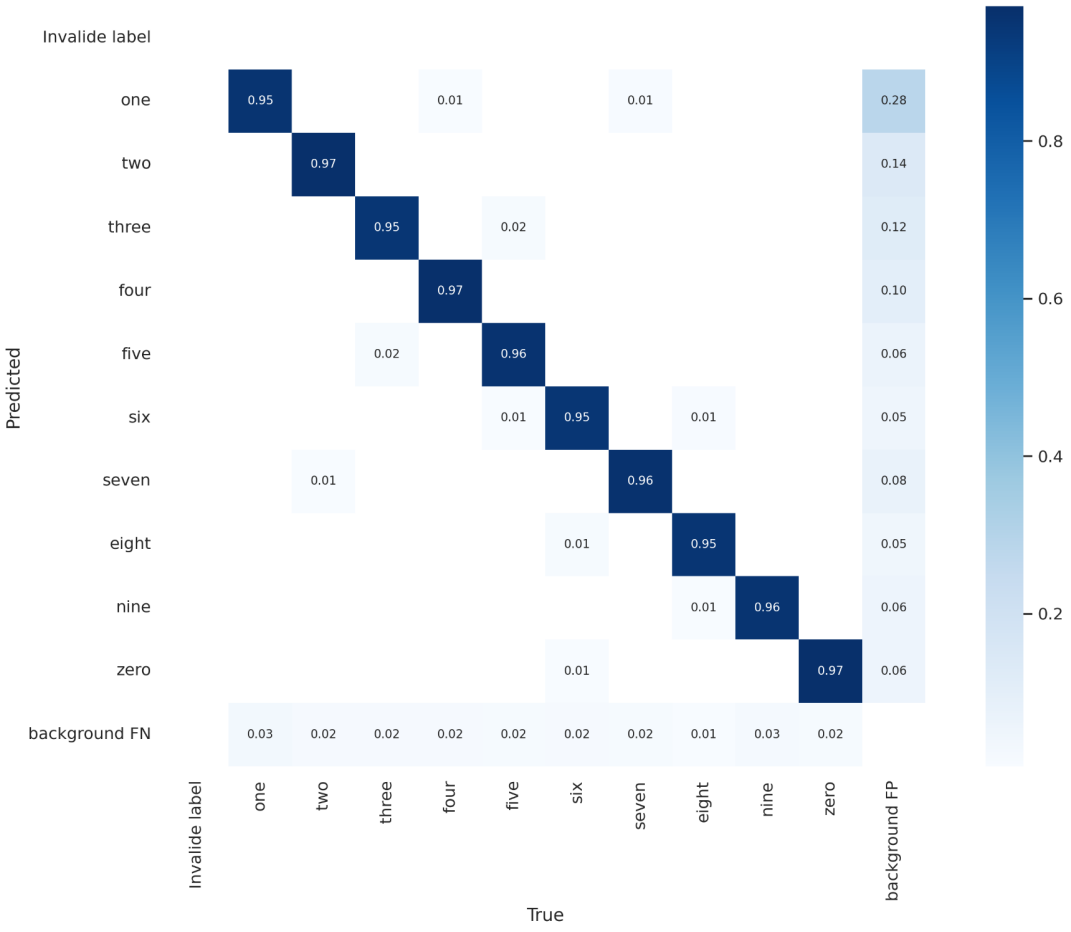
The model architecture I used is as follows:

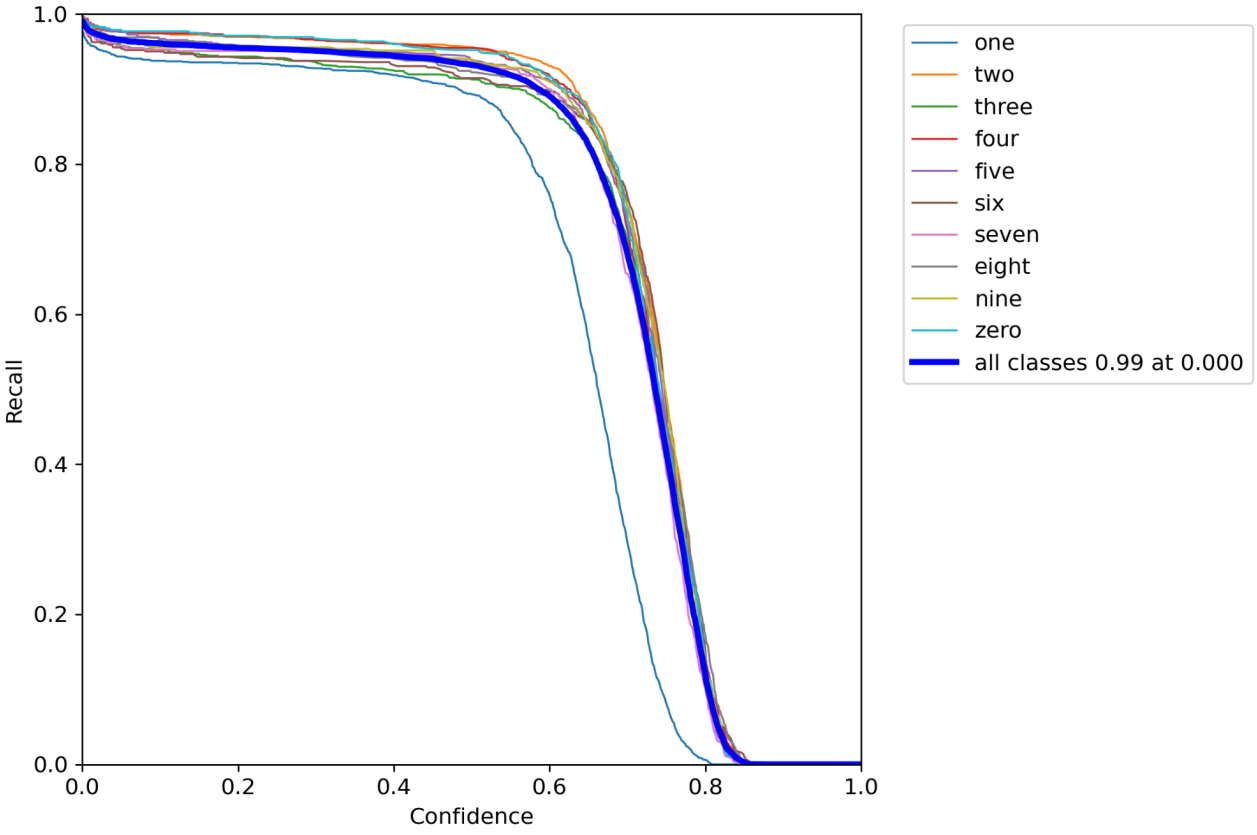
| parameters | value | explanation |
|---|---|---|
| Pretrained model | Yolov5l | Pretrained on COCO |
| batch_size | 16 | Batch size |
| epoches | 80 | Epochs to train |
| lr0 | 0.0032 | initial learning rate |
| lrf | 0.12 | final OneCycleLR learning rate (lr0 * lrf) |
| momentum | 0.843 | SGD momentum |
| weight_decay | 0.00036 | optimizer weight decay 5e-4 |
| box | 0.0296 | box loss gain |
| cls | 0.243 | cls loss gain |
| cls_pw | 0.631 | cls BCELoss positive_weight |
| obj | 0.301 | obj loss gain (scale with pixels) |
| obj_pw | 0.911 | obj BCELoss positive_weight |
| degrees | 0.373 | image rotation (+/- deg) |
| translate | 0.245 | image translation (+/- fraction) |
| scale | 0.898 | image scale (+/- gain) |
| flipud | 0.0 | image flip up-down (probability) |
| fliplr | 0.0 | image flip left-right (probability) |
| mosaic | 1.0 | image mosaic (probability) |

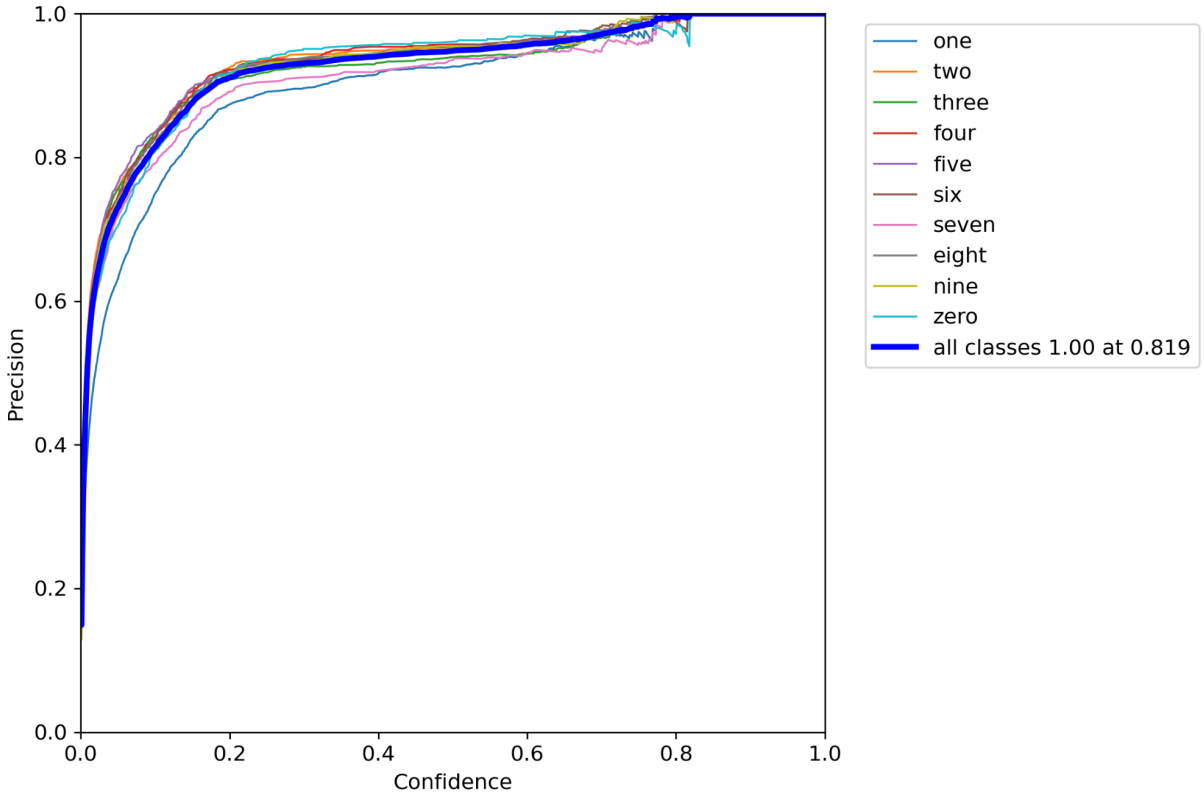
# Summary

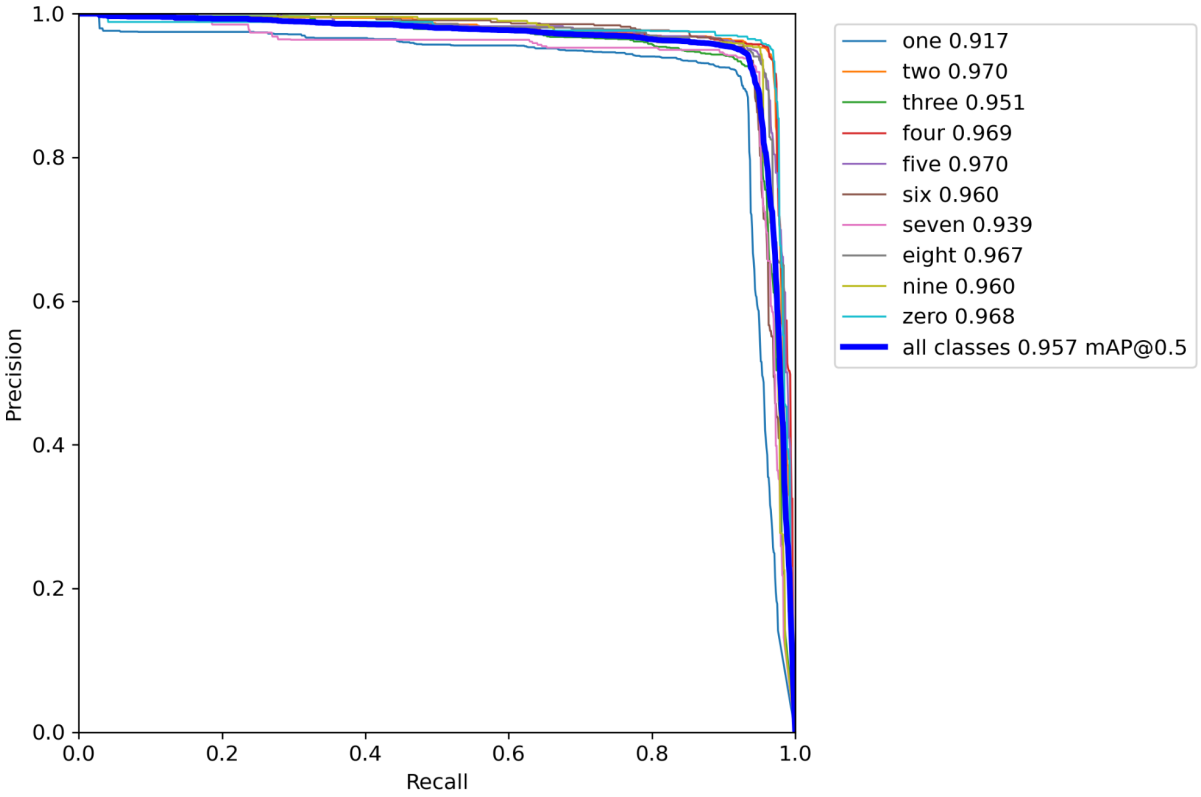
In conclusion, during HW2, I learned to fine tun existing objection detection model, I also learned to deploy my model on Colab environment. The accuracy and timing of my results are acceptable.

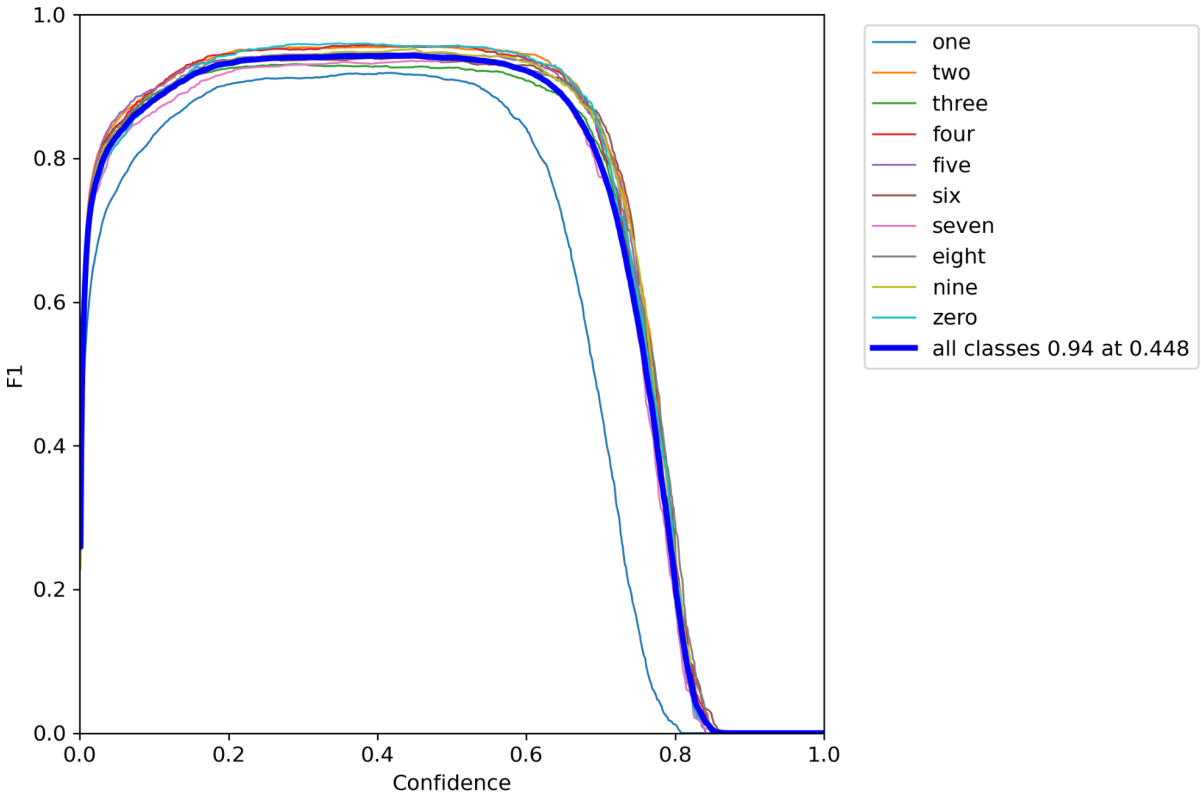
The Confusion Matrix and P_Curves, R_Curve and F1_Curve are shown below.




Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
one
two
three
four
five
six
seven
eight
nine
zero
all classes 0.99 at 0.000


Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
one
two
three
four
five
six
seven
eight
nine
zero
all classes 1.00 at 0.819


Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
one 0.917
two 0.970
three 0.951
four 0.969
five 0.970
six 0.960
seven 0.939
eight 0.967
nine 0.960
zero 0.968
all classes 0.957 mAP@0.5


F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
one
two
three
four
five
six
seven
eight
nine
zero
all classes 0.94 at 0.448