Selected Topics in Visual Recognition using Deep Learning
HW2 Report: Street View House Numbers detection

310706004 Hsin-Ying Lien

I. Introduction

In this assignment, I have tried to detect the numbers with two object detection tools — MMDetection and YOLOv5. Finally, I used YOLOv5 and fine-tune it based on weight yolov5m.pt to accomplish the object detection on the given number dataset. It reached a final mean Average Precision about 0.40261 on the test dataset. The inference time is shown below (The code is shown in part III).

```
Speed: 0.3ms pre-process, 39.4ms inference, 1.3ms NMS per image at shape (1, 3, 640, 640)

Inference time per image:  0.10491699934005737
```

Code GitHub link: https://github.com/angelalien/VRDL-HW2
Dataset: Street View House Numbers Dataset (training: 33,402, test: 13,068)

II. Methodology

I implement YOLOv5 by following the steps mentioned in a website(YOLOv5 實現目標檢測) . I first downloaded the YOLOv5 git file, and made some changes with some files in it. Part of the file structure in *VRDL-HW2* (the git file downloaded) is shown below.

```
|— data
|    |— images       #contains all train images and digitStruct.mat
|    |— ImageSets    #contains train-validation-test spilt text files
|    |— labels       #contains annotation text files
```

The file structure in *ImageSets* is shown below.

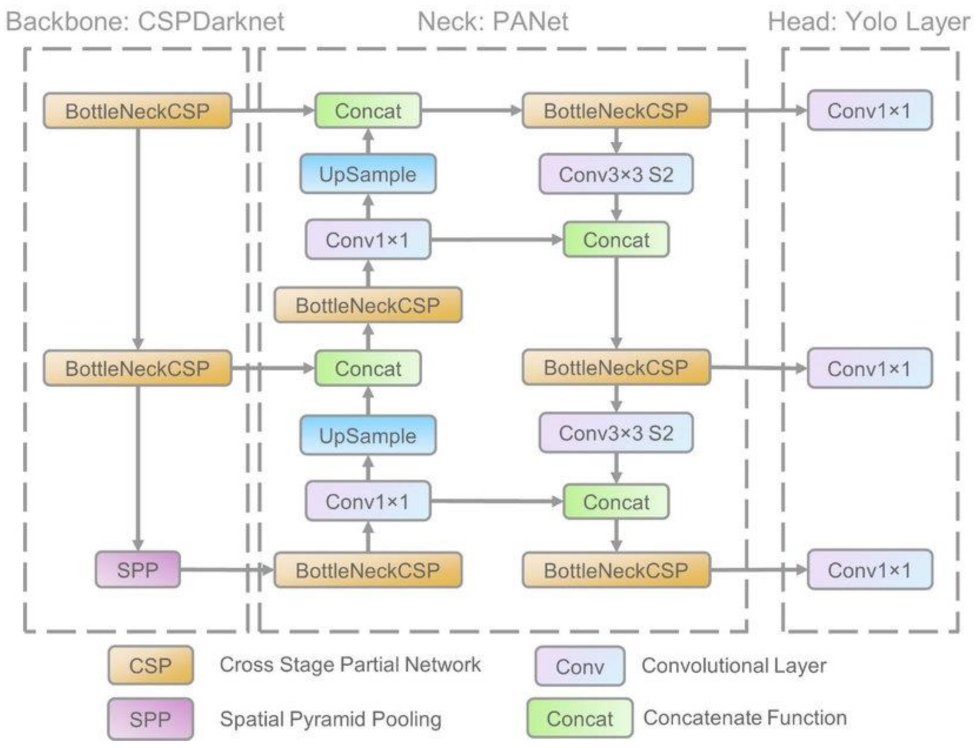
```
|— ImageSets
|    |— train.txt
|    |— val.txt
|    |— trainval.txt
|    |— test.txt
```

1. Data Pre-process

1) Transform annotation file format:
   I transformed the *.mat* file into *{image_id}.txt* file in the *mat_label.py* file. Each text file would save the label and bounding box information of one image. These text files were saved in *labels* file.
2) Prepare train, validation and test data:
   I wrote the image file path into three main text file—*train.txt*, *val.txt*, *and test.txt*, saving them in *ImageSets* file. The proportion of training, validation, and testing data is 8:1:1.
3) Modify dataset setting file:
   I copied *coco.yaml* file, renaming it as *SVHN.yaml* file. Then I modified the file path of training, validation, testing file and the information of classes.
4) Modify model setting file:
   I modified the classes number into 10 in *yolov5m.yaml* file.

2. Model architecture

The network architecture of Yolov5. It consists of three parts: (1) Backbone: CSPDarknet, (2) Neck: PANet, and (3) Head: Yolo Layer. The data are first input to CSPDarknet for feature extraction, and then fed to PANet for feature fusion. Finally, Yolo Layer outputs detection results (class, score, location, size).



3. **Hyperparameters**

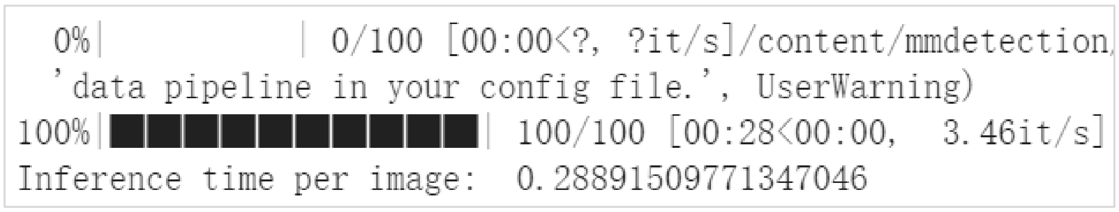
batch size = 16
epochs = 10
initial learning rate=0.01, momentum=0.937, weight_decay=0.0005
warmup_epochs=3, warmup_momentum=0.8, warmup_bias_lr=0.1
detector: confidence threshold=0.2

III. Experimental Results and Observations

I have tried two different object detection tools for this assignment task.

1) MMDetection

In the beginning, I transform the *.mat* file into coco format file, putting it into the MMDetection detector with Faster R-CNN backbone and finetuning it. I found that it doesn't improve the performance to reduce the learning rate to 0.01 from the default 0.02. However, it took me more than five hours to train for an epoch with my computer using MMDetection. I think it is because the images are resized to large scale during preprocess in default setting, which provides more information of the image. But I only got mean Average Precision about 0.3478, due to only one training epoch. The detector's inference time is about 0.289 per image.



```
  0%|          | 0/100 [00:00<?, ?it/s]/content/mmdetection
  'data pipeline in your config file.', UserWarning)
100%|██████████| 100/100 [00:28<00:00,  3.46it/s]
Inference time per image:  0.28891509771347046
```

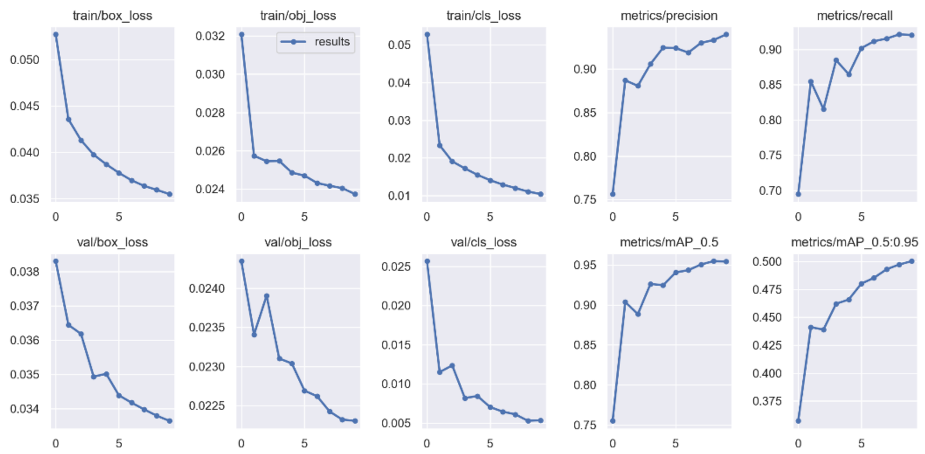
2) YOLOv5

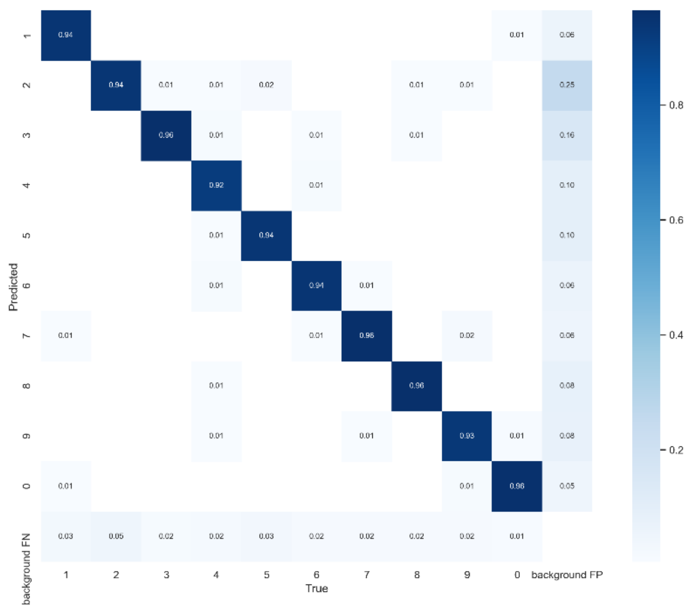
With the constraint of my hardware equipment, I decided to use YOLOv5 as my detection tool. Since YOLO is a lighter detector, I could spend the same time to train the data for 10 epochs. I tried two pretrained weight—yolov5s.pt and yolov5m.pt.

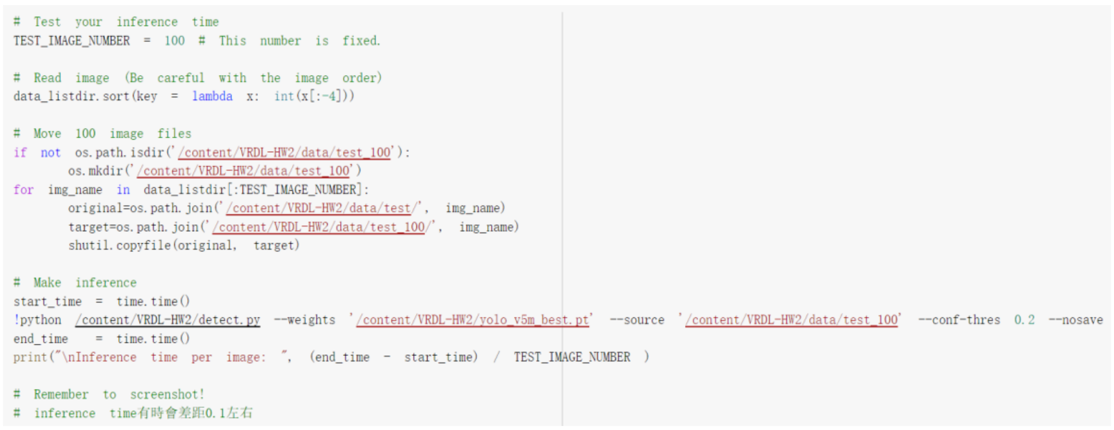
Pretrained Weight	Hyperparameter	mAP	Speed(s/image)
yolov5m.pt	epoch=10	0.402611	0.1069
yolov5s.pt	epoch=10	0.3963	0.085
yolov5s.pt	epoch=20	0.401366	0.085

From the above result, it is normal that the performance of the finetuned model using yolov5m.pt pretrained weight is a little bit better than the one using yolov5s.pt pretrained weight, although the first one's inference time is slower. We can also find that model using yolov5s.pt has to be trained for longer time to get almost the same performance as the one using yolov5m.pt.

Finally, I chose the model using yolov5m.pt as the best model. The training process plots of loss, precision, recall and mean Average Precision are shown below. The model began to converge after the fifth epoch. From the confusion matrix, we can also know that the model learns well. And the screenshot of inference speed is shown below







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

# Read image (Be careful with the image order)
data_listdir.sort(key = lambda x: int(x[:-4]))

# Move 100 image files
if not os.path.isdir('/content/VRDL-HW2/data/test_100'):
       os.mkdir('/content/VRDL-HW2/data/test_100')
for img_name in data_listdir[:TEST_IMAGE_NUMBER]:
       original=os.path.join('/content/VRDL-HW2/data/test/', img_name)
       target=os.path.join('/content/VRDL-HW2/data/test_100/', img_name)
       shutil.copyfile(original, target)

# Make inference
start_time = time.time()
!python /content/VRDL-HW2/detect.py --weights '/content/VRDL-HW2/yolo_v5m_best.pt' --source '/content/VRDL-HW2/data/test_100' --conf-thres 0.2 --nosave
end_time   = time.time()
print("\nInference time per image: ", (end_time - start_time) / TEST_IMAGE_NUMBER )

# Remember to screenshot!
# inference time有時會差距0.1左右
```

```
Speed: 0.3ms pre-process, 39.4ms inference, 1.3ms NMS per image at shape (1, 3, 640, 640)

Inference time per image:  0.10491699934005737
```

3) Comparison

With the results above, I think that YOLOv5 is better than MMDetection for handling this number detection task. Since MMDetection will predict lots of bounding boxes and classification result for each number, it takes more time to train and detect. If I want to train a model in a short time with low level graphic processing unit, I have to resize the image to small scale but may get worse result with less information. YOLO instead predicts only one bounding box and number result per number, which is faster and efficient. I could spend less time to inference the model prediction and transform it into COCO format. Also, the output performance is great. As a result, for those people lacking in high level graphic processing unit and time, I will recommend using YOLO at first.

IV. Summary

Nowadays, it's easy for us to implement pretrained models and fine-tune it to deal with object detection tasks. However, we have to consider our available GPU devices while training with large amount of training data. At this moment, proper choose of object detection tools plays an important role in avoiding this situation if we are not equipped with advanced hardware environment. Otherwise, we have to change the data augmentation methods to speed up the training process, such as reduce the image size. As a result, data preprocessing and detector selection are important for training time. And the second one also influences the inference time.

V. Reference

h5py, access data in Datasets in SVHN
https://stackoverflow.com/questions/41176258/h5py-access-data-in-datasets-in-svhn/41264930

YOLOv5 實現目標檢測
https://tw511.com/a/01/29504.html

YOLOv5 Model Architecture
https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet_fig1_349299852