Homework 2 - Object Detection

310581044 陳柏勳

Experiment Setup

python 版本是3.7.13

系統使用windows10

安裝環境

使用anaconda 重新開始創建一個新環境使用conda list之結果將使用套

件與套件版本如下：

# Name	Version	Build	Channel
absl-py	1.3.0	pypi_0	pypi
argon2-cffi	21.3.0	pyhd3eb1b0_0	
argon2-cffi-bindings	21.2.0	py37h2bbff1b_0	
attrs	22.1.0	pypi_0	pypi
backcall	0.2.0	pyhd3eb1b0_0	
beautifulsoup4	4.11.1	py37haa95532_0	
blas	1.0	mkl	
bleach	4.1.0	pyhd3eb1b0_0	
brotlipy	0.7.0	py37h2bbff1b_1003	
ca-certificates	2022.10.11	haa95532_0	
cachetools	5.2.0	pypi_0	pypi
certifi	2022.9.24	py37haa95532_0	
cffi	1.15.1	py37h2bbff1b_0	
charset-normalizer	2.0.4	pyhd3eb1b0_0	
colorama	0.4.5	py37haa95532_0	
cpuonly	1.0	0	pytorch
cryptography	37.0.1	py37h21b164f_0	
cudatoolkit	11.3.1	h59b6b97_2	
cxxfilt	0.3.0	pypi_0	pypi
cycler	0.11.0	pypi_0	pypi

cython	0.29.32	pypi_0	pypi
debugpy	1.5.1	py37hd77b12b_0	
decorator	5.1.1	pyhd3eb1b0_0	
defusedxml	0.7.1	pyhd3eb1b0_0	
entrypoints	0.4	py37haa95532_0	
fftw	3.3.9	h2bbff1b_1	
flatbuffers	22.9.24	pypi_0	pypi
fonttools	4.37.4	pypi_0	pypi
freetype	2.10.4	hd328e21_0	
google-auth	2.13.0	pypi_0	pypi
google-auth-oauthlib	0.4.6	pypi_0	pypi
grpcio	1.50.0	pypi_0	pypi
icc_rt	2022.1.0	h6049295_2	
idna	3.4	py37haa95532_0	
importlib-metadata	5.0.0	pypi_0	pypi
importlib_metadata	4.11.3	hd3eb1b0_0	
importlib_resources	5.2.0	pyhd3eb1b0_1	
iniconfig	1.1.1	pypi_0	pypi
intel-openmp	2021.4.0	haa95532_3556	
ipykernel	6.15.2	py37haa95532_0	
ipython	7.31.1	py37haa95532_1	
ipython_genutils	0.2.0	pyhd3eb1b0_1	
jedi	0.18.1	py37haa95532_1	
jinja2	3.0.3	pyhd3eb1b0_0	
joblib	1.1.1	py37haa95532_0	
jpeg	9b	hb83a4c4_2	
jsonschema	4.16.0	py37haa95532_0	
jupyter_client	7.3.5	py37haa95532_0	
jupyter_core	4.11.1	py37haa95532_0	
jupyterlab_pygments	0.1.2	py_0	
kiwisolver	1.4.4	pypi_0	pypi
lerc	3.0	hd77b12b_0	
libdeflate	1.8	h2bbff1b_5	
libpng	1.6.37	h2a8f88b_0	
libsodium	1.0.18	h62dcd97_0	
libtiff	4.4.0	h8a3f274_0	
libuv	1.40.0	he774522_0	
libwebp	1.2.4	h2bbff1b_0	

libwebp-base	1.2.4	h2bbff1b_0	
loguru	0.6.0	pypi_0	pypi
lz4-c	1.9.3	h2bbff1b_1	
markdown	3.4.1	pypi_0	pypi
markupsafe	2.1.1	py37h2bbff1b_0	
matplotlib	3.5.3	pypi_0	pypi
matplotlib-inline	0.1.6	py37haa95532_0	
mistune	0.8.4	py37hfa6e2cd_1001	
mkl	2021.4.0	haa95532_640	
mkl-service	2.4.0	py37h2bbff1b_0	
mkl_fft	1.3.1	py37h277e83a_0	
mkl_random	1.2.2	py37hf11a4ad_0	
nbclient	0.5.13	py37haa95532_0	
nbconvert	6.4.4	py37haa95532_0	
nbformat	5.5.0	py37haa95532_0	
nest-asyncio	1.5.5	py37haa95532_0	
ninja	1.10.2.4	pypi_0	pypi
ninja-base	1.10.2	h6d14046_5	
notebook	6.4.12	py37haa95532_0	
numpy	1.21.5	py37h7a0a035_3	
numpy-base	1.21.5	py37hca35cd5_3	
oauthlib	3.2.2	pypi_0	pypi
onnx	1.8.1	pypi_0	pypi
onnx-simplifier	0.3.5	pypi_0	pypi
onnxoptimizer	0.3.1	pypi_0	pypi
onnxruntime	1.8.0	pypi_0	pypi
opencv-python	4.6.0.66	pypi_0	pypi
openssl	1.1.1q	h2bbff1b_0	
packaging	21.3	pyhd3eb1b0_0	
pandocfilters	1.5.0	pyhd3eb1b0_0	
parso	0.8.3	pyhd3eb1b0_0	
pickleshare	0.7.5	pyhd3eb1b0_1003	
pillow	9.2.0	py37hdc2b20a_1	
pip	22.3	pypi_0	pypi
pkgutil-resolve-name	1.3.10	py37haa95532_0	
pluggy	1.0.0	pypi_0	pypi
prometheus_client	0.14.1	py37haa95532_0	
prompt-toolkit	3.0.20	pyhd3eb1b0_0	

protobuf	3.19.6	pypi_0	pypi
psutil	5.9.0	py37h2bbff1b_0	
py	1.11.0	pypi_0	pypi
pyasn1	0.4.8	pypi_0	pypi
pyasn1-modules	0.2.8	pypi_0	pypi
pycocotools	2.0.5	pypi_0	pypi
pycparser	2.21	pyhd3eb1b0_0	
pygments	2.11.2	pyhd3eb1b0_0	
pyopenssl	22.0.0	pyhd3eb1b0_0	
pyparsing	3.0.9	py37haa95532_0	
pyrsistent	0.18.0	py37h196d8e1_0	
pysocks	1.7.1	py37_1	
pytest	7.1.3	pypi_0	pypi
python	3.7.13	h6244533_1	
python-dateutil	2.8.2	pyhd3eb1b0_0	
python-fastjsonschema	2.16.2	py37haa95532_0	
pytorch-mutex	1.0	cpu	pytorch
pywin32	302	py37h2bbff1b_2	
pywinpty	2.0.2	py37h5da7b33_0	
pyyaml	6.0	pypi_0	pypi
pyzmq	23.2.0	py37hd77b12b_0	
requests	2.28.1	py37haa95532_0	
requests-oauthlib	1.3.1	pypi_0	pypi
rsa	4.9	pypi_0	pypi
scikit-learn	1.0.2	py37hf11a4ad_1	
scipy	1.7.3	py37h7a0a035_2	
send2trash	1.8.0	pyhd3eb1b0_1	
setuptools	63.4.1	py37haa95532_0	
six	1.16.0	pyhd3eb1b0_1	
soupsieve	2.3.2.post1	py37haa95532_0	
sqlite	3.39.3	h2bbff1b_0	
tabulate	0.9.0	pypi_0	pypi
tensorboard	2.10.1	pypi_0	pypi
tensorboard-data-server	0.6.1	pypi_0	pypi
tensorboard-plugin-wit	1.8.1	pypi_0	pypi
terminado	0.13.1	py37haa95532_0	
testpath	0.6.0	py37haa95532_0	
thop	0.1.1-2209072238	pypi_0	pypi

threadpoolctl	2.2.0	pyh0d69192_0	
tk	8.6.12	h2bbff1b_0	
tomli	2.0.1	pypi_0	pypi
torch	1.12.0+cu116	pypi_0	pypi
torchaudio	0.12.0+cu116	pypi_0	pypi
torchvision	0.13.0+cu116	pypi_0	pypi
tornado	6.2	py37h2bbff1b_0	
tqdm	4.64.1	pypi_0	pypi
traitlets	5.1.1	pyhd3eb1b0_0	
typing-extensions	4.3.0	py37haa95532_0	
typing_extensions	4.3.0	py37haa95532_0	
urllib3	1.26.11	py37haa95532_0	
vc	14.2	h21ff451_1	
vs2015_runtime	14.27.29016	h5e58377_2	
wcwidth	0.2.5	pyhd3eb1b0_0	
webencodings	0.5.1	py37_1	
werkzeug	2.2.2	pypi_0	pypi
wheel	0.37.1	pyhd3eb1b0_0	
win32-setctime	1.1.0	pypi_0	pypi
win_inet_pton	1.1.0	py37haa95532_0	
wincertstore	0.2	py37haa95532_2	
winpty	0.4.3	4	
xz	5.2.6	h8cc25b3_0	
yolox	0.3.0	dev_0	<develop>
zeromq	4.3.4	hd77b12b_0	
zipp	3.9.0	pypi_0	pypi
zlib	1.2.12	h8cc25b3_3	
zstd	1.5.2	h19a0ad4_0	

環境建構方式

將yolox、apex從github上clone下來

安裝yolox

進入yolox資料夾

pip3 install -U pip

pip3 install -r requirements.txt

python3 setup.py develop

此處因為cuda是11.6版為符合使用下列指令

```
conda install pytorch==1.12.0 torchvision==0.13.0
torchaudio==0.12.0 cudatoolkit=11.6 -c pytorch -c conda-forge
```

安裝pytorch torchvision torchaudio cudatoolkit

參考Previous PyTorch Versions | PyTorch

這些版本比yolox提供安裝版本還新

安裝apex(github網址：NVIDIA/apex: A PyTorch Extension: Tools for easy mixed precision and distributed training in Pytorch (github.com)**)**

進入apex資料夾

pip install -r requirements.txt

python setup.py install

安裝pycocotools

pip install pycocotools

使用anaconda安裝 scikit-image

下載pretrained weight

將yolo_m weight載到github code所在同個資料夾(實驗測試時有用yolo_s weight但由於最後best model是用yolo_m故把yolo_s刪掉了)

參考YOLOX实战：超详细！手把手教你使用YOLOX进行物体检测（附数据集）-云社区-华为云 (huaweicloud.com)

yolox 训练自己的数据集 （COCO格式）_zxxRobot的博客-CSDN博客_yolo训练coco数据集

註：APEX是英偉達開源的，完美支持PyTorch框架，用於改變數據格式來減小模型顯存佔用的工具。其中最有價值的是amp（Automatic Mixed Precision），將模型的大部分操作都用Float16數據類型測試，一些特別操作仍然使用Float32。並且用戶僅僅通過三行代碼即可完美將自己的訓練代碼遷移到該模型。實驗證明，使用Float16作為大部分操作的數據類型，並沒有降低參數，在一些實驗中，反而由於可以增大Batch size，帶來精度上的提升，以及訓練速度上的提升。

Apex部分由於數篇yolox安裝教學都有安裝所以就跟著安裝並附在yolox資料夾裡，但似乎不會影響在訓練推論時的指令，可能是在訓練時會自動啟用

■ Data pre-process, Model architecture, Hyperparameters,...

Data pre-process部分主要只有做轉換成coco格式詳細部分在後面說明

Hyperparameters

./exps/example/custom/yolox_m.py

```
class Exp(MyExp):
    def __init__(self):
        super(Exp, self).__init__()
        self.depth = 0.67
        self.width = 0.75
        self.exp_name =
os.path.split(os.path.realpath(__file__))[1].split(".")[0]
        # Define yourself dataset path
        self.data_dir = "datasets/final"
        self.train_ann = "train.json"
        self.val_ann = "val.json"
        self.num_classes = 1
        self.max_epoch = 100
        #self.max_epoch = 300
        self.data_num_workers = 0
        self.eval_interval = 1
```

這次epoch我都先設100看到loss約跑10次都沒再下降就提早結束，後面發現約設為30~40次對於本任務就差不多ok了

Data_num_workers代表多進程執行數量，由於我的設備關係，只能用1進程，故設為0

Eval_inerval=1代表每次epoch訓練完會對val做eval

Dataset_path部分

放在datasets/final裡有annotations train2107 val2017，train2017就是原本的train資料夾，val2017就是原本的val資料夾，annotations放有train.json val.json 代表是已轉成coco格式的檔案

./yolox/data/datasets/coco_classes.py

Coco class改成只有car

Other parameter

由於這次使用的yolox_m參數量相對多，是在我自己3060顯卡的筆電上跑，使用yolo_l或以上的weight，我的筆電無法負荷，在執行訓練時是使用下面指令

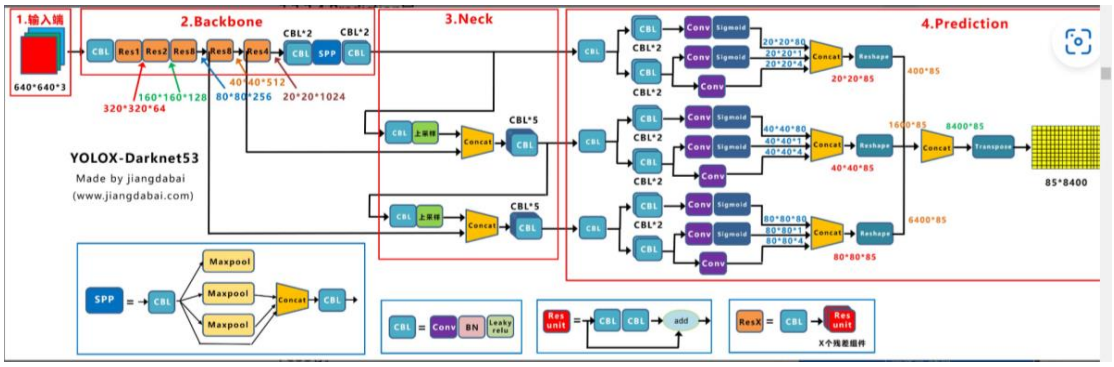
```
python tools/train.py -f exps/example/custom/yolox_m.py -d 0 -b 8 --fp16 -o -c yolox_m.pth
```

-d 使用多少張顯卡訓練，-b batch size 大小，-o 訓練時優先搶占 gpu 資源，–fp16 是否開啟半精度訓練，-c checkpoint path，Batch 在>8 情況下都會 out of cuda，故設定為 8

其餘hyperparameters都是用預設的數值，因為原code提供yolox已將大部分表現最好的hyperparameters當成預設，隨意改動可能反而會降低表現，故大致上都遵循原本參數。

Model architecture

這次是使用作業說明提供的yolox架構，並以yolox_m相關ckpt和weight做訓練，而主要有修改的架構在下圖中的backbone部分，而其code相關在yolo_pafpn.py做修改



○ Brief explain your code

■ SE module

SE module簡單的說，主要是學習了channel之間的相關性，篩選出了針對通道的attention，通過對convolution的到的feature map進行處理，得到一個和通道數一樣的一維向量作為每個通道的評價分數，然後將改分數分別施加到對應的通道上，加以改進成效

對於添加方法與成果SE module可以加到一個network block結束的位置，進行一個信息refine ，如resnet50,resnext50，bn-inception等網絡。通過添加SE module，能使模型提升0.5-1.5%,效果

實行方式都在/yolox/models/yolo_pafpn.py上操作，基本上直接套用作業介紹的code先寫成一個block

```
class SE(nn.Module):
    def __init__(self, channel, ratio=16):
        super(SE, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
                nn.Linear(channel, channel // ratio, bias=False),
                nn.ReLU(inplace=True),
                nn.Linear(channel // ratio, channel, bias=False),
                nn.Sigmoid()
        )
    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
```

```
        return x * y
```

然後在

```
class YOLOPAFPN(nn.Module):
```

中結尾加入

```
        self.SE_1 = SE(int(in_channels[2] * width))   #對應 dark5 輸出 1024
channel

        self.SE_2 = SE(int(in_channels[1] * width)) #對應 dark4 輸出 512
channel

        self.SE_3 = SE(int(in_channels[0] * width)) #對應 dark3 輸出 256
channel
```

代表符合不同darknet output channel數量的se module方便後面插入，

緊接之後在

```
def forward(self, input):
```

有features 陣列

```
features = [out_features[f] for f in self.in_features]

        [x2, x1, x0] = features
```

X2 x1 x0就是輸出通道的features然後就可以視想加入的se module對X2 x1 x0

做操作，如下面所示，使用一次self.se即代表加入一個se module

```
# x0 = self.SE_1(x0)
```

```
        #x0 = self.SE_1(x0)
        #x0 = self.SE_1(x0)
        #x0 = self.SE_1(x0)
        #x0 = self.SE_1(x0)
      #x1 = self.SE_2(x1)
        x2 = self.SE_3(x2)
        x2 = self.SE_3(x2)
        x2 = self.SE_3(x2)
```

這些通過se module後的features 會依序丟入fpn作後續操作，其餘部分yolox因為沒有修改直接套用，就沒有解釋，data前處理和對data output做修改在other部分做說明

How to start

將yolo label轉換成coco格式

具體作法後面有說明，進入dataset/hw裡，按照說明改好資料夾名字和位置執行，生成之annotations和train2017(即train)、val2017(即val)放在dataset/final裡

training / inference command line

如何開始訓練

進入到 yolox github 已 clone 下來之資料夾後使用指令

python tools/train.py -f exps/example/custom/yolox_m.py -d 0 -b 8 --fp16 -o -c yolox_m.pth 即會開始訓練，並將結果存在 /YOLOX_outputs/yolox_m 資料夾裡，

如何計算 map 值

將 val dataset 放在 assets 資料夾，執行

python tools/demo.py image -f exps/example/custom/yolox_m.py -c YOLOX_outputs/yolox_m/best_ckpt.pth --path assets/val/ --conf 0.25 --nms 0.5 --tsize 640 --save_result --device gpu

生成 val 的 output 結果

再把 output 結果複製到 map_cal/detections

Val_labels 資料夾內容複製到 map_cal/groundtruth_rel

在 map_cal 中執行指令

python pascalvoc.py -gt groundtruths_rel/ -gtformat xywh -gtcoords rel -det detections/ -detformat xyrb -imgsize 1920,1080 -t 0.85 -np

則結果會在 map_cal/results 資料夾顯示出圖表和各圖片比較數值

testing data 放置地方

將要 inference 的圖片以資料夾形式放在 assets 資料夾(以 test 為例將整個 test 資料夾放進 assets)

testing result reproduced

執行指令

python tools/demo.py image -f exps/example/custom/yolox_m.py -c YOLOX_outputs/yolox_m/best_ckpt.pth --path assets/test/ --conf 0.25 --nms 0.5 --tsize 640 --save_result --device gpu

結果會存在/YOLOX_outputs/yolox_m/vis_res 根據執行當下時間命名的資料夾哩，裡面即有各 test 圖片 inference 出的 txt 檔

■ If you used code from GitHub, provide reference

Reference

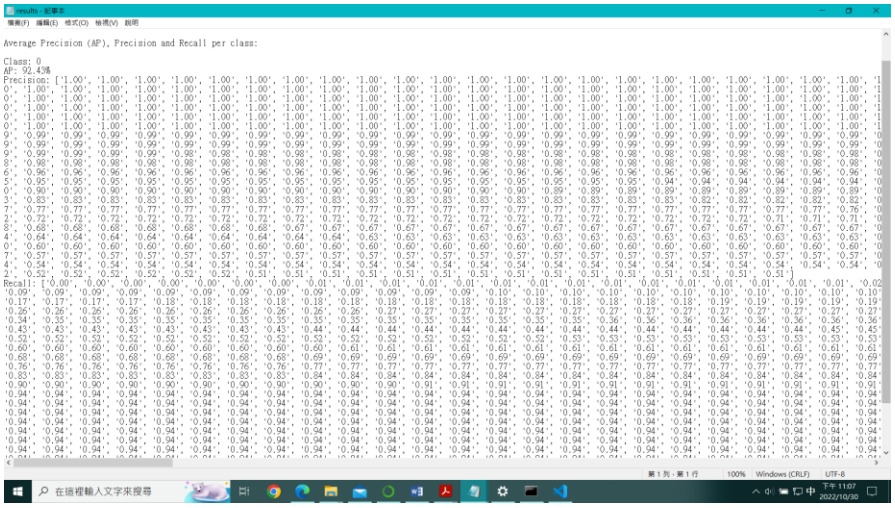
1.yolox Megvii-BaseDetection/YOLOX: YOLOX is a high-performance anchor-free YOLO, exceeding yolov3~v5 with MegEngine, ONNX, TensorRT, ncnn, and OpenVINO supported. Documentation: https://yolox.readthedocs.io/ (github.com)

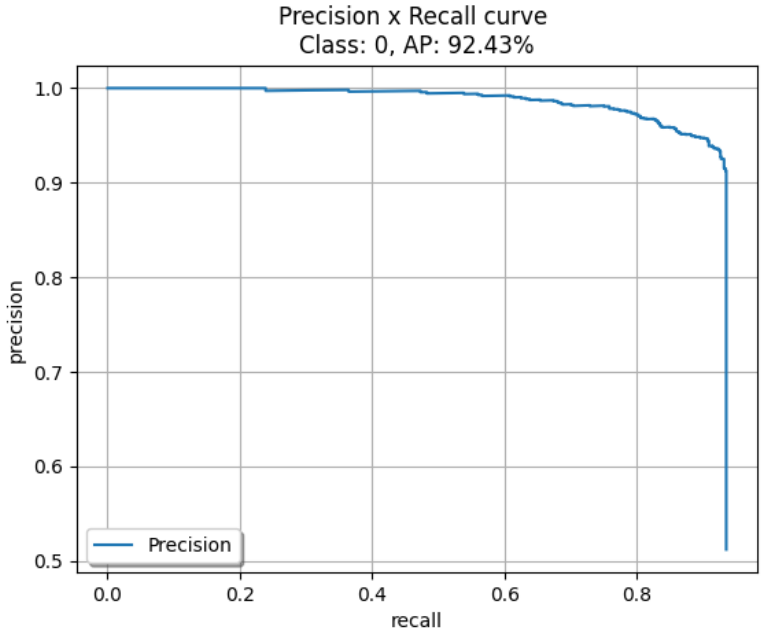
2.yolo to coco Yolo-to-COCO-format-converter/main.py at master · Taeyoung96/Yolo-to-COCO-format-converter (github.com)

3.檢測map工具rafaelpadilla/Object-Detection-Metrics: Most popular metrics used to evaluate object detection algorithms. (github.com)

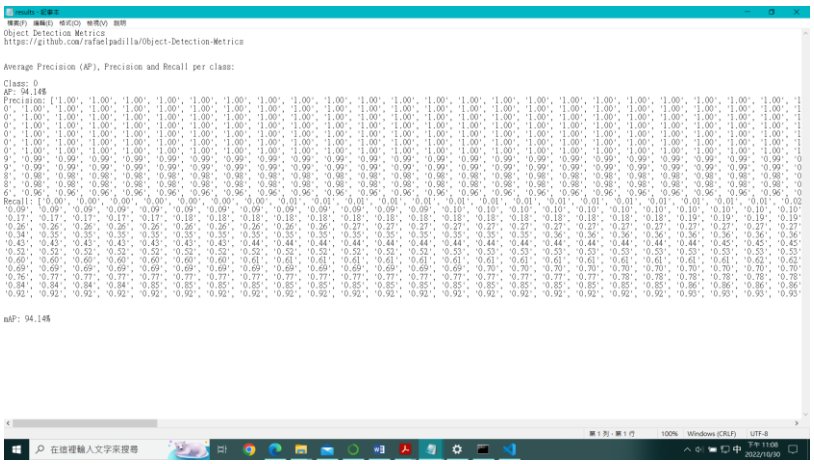
○ Screenshot your validation results on your two models (with / without SE module)(後面有針對完整此表格完整截圖)

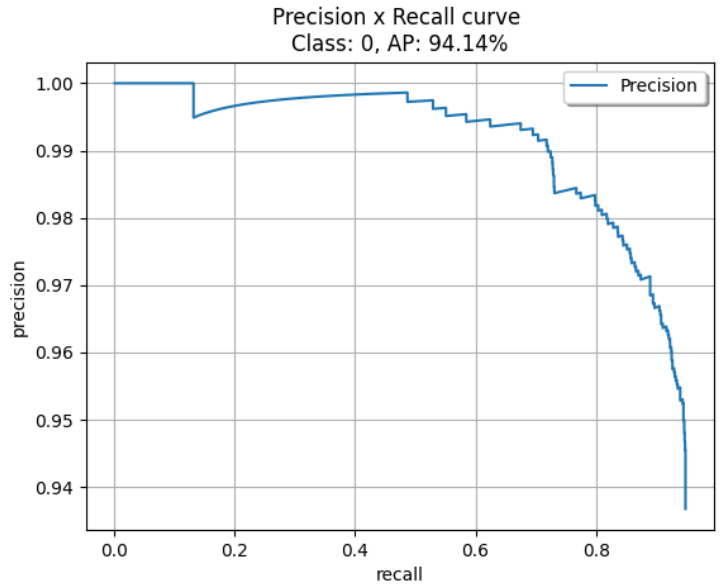
1. without SE module





2. with SE module



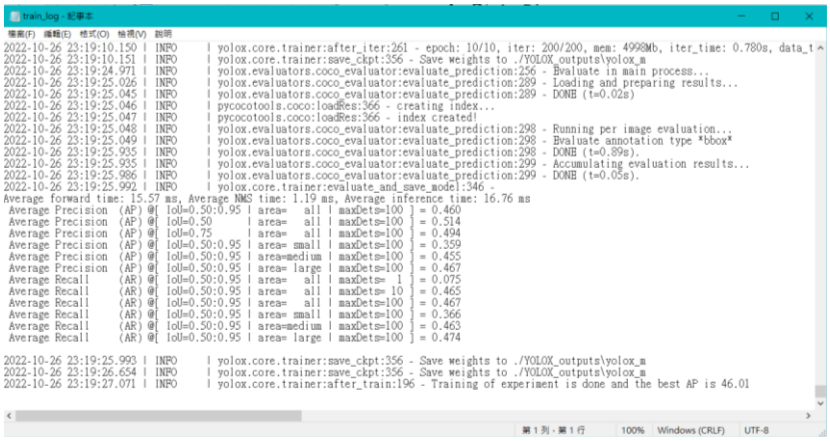


○ Discussion

■ Problems you encountered

1.

第一個我卡很久的問題是在轉換格式那裏，對yolo轉coco格式，我一開始把yolox x y代表的是bbox中心座標和coco 的x y代表的是bbox左上座標搞混了，所以開始訓練時不管如何訓練，改多少參數都無法成功到達標準而且還不到50%正確率，以下附上相關截圖，



而我想也不會是訓練次數不夠問題因為實在跟一般正常正確率差太大，而我又做了生成圖片測試看結果，如下圖

Bbox幾乎都有正確框出車子，代表模型是有抓到，所以我判斷這裡應該是bbox的gt有錯，也就是我轉換格式過程有錯所以正確率很低，接著發現我把coco格式的xy設定成bbox中心座標了，修改成左上座標過後再重train就成功到達要求了。

2.

主要是執行code上遇到的一些困難，有些不知道是原code提供者是在linux環境上跑或是不同編譯器問題，但這些主要網路上都有相對應解決方法像是出現 subprocess.CalledProcessError: Command '['where', 'cl']' returned non-zero exit status 1

解決方法對 yolox/evaluators/coco_evaluator.py 修改

try:

    from yolox.layers import COCOeval_opt as COCOeval

except ImportError:

    from pycocotools.cocoeval import COCOeval

改成只有

from pycocotools.cocoeval import COCOeval

網路上說這是python轉換成c++出現的位置編碼錯誤的問題，可能是跟visual studio某個套件要安裝有關，但我這次是用anaconda虛擬環境跑所以算是不太相關的軟件引發錯誤

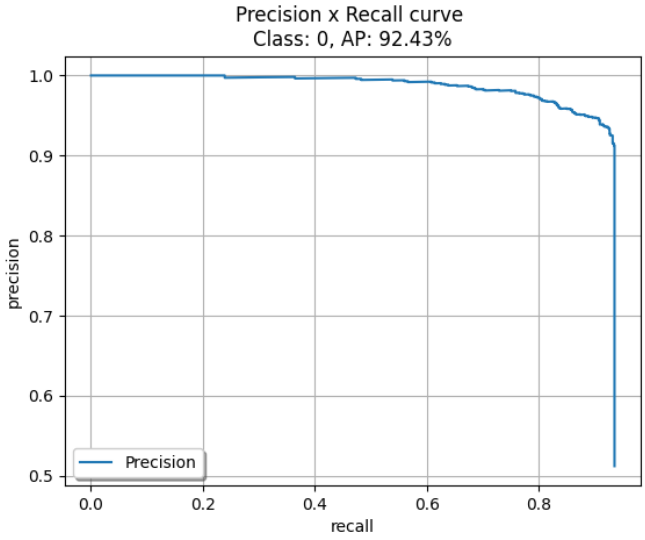
3.

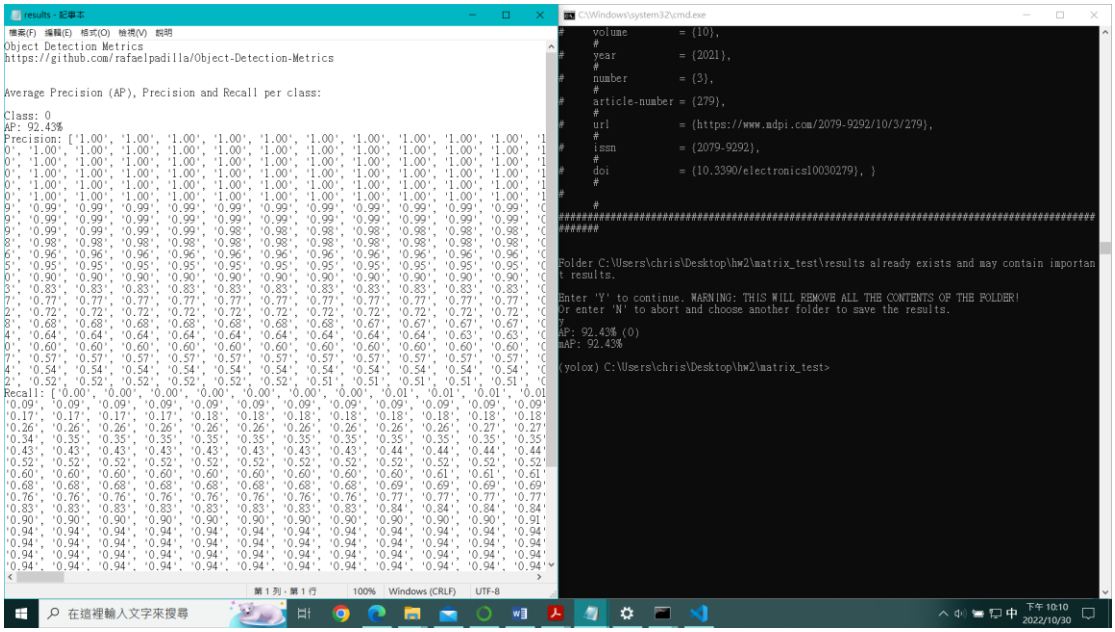
主要是硬體設備不夠好的訓練問題，目前我跑 30 次的 epoch 大概就需要 2~3 小時，而且訓練過程常常會自己停下來，需要我再按一下螢幕才能讓顯卡運轉，導致訓練次數無法太多，實驗不行做太多測試，可能是顯卡過熱或是我在-o 訓練時優先搶占 gpu 資源時要設 true 讓顯卡能吃多點效能跑快點

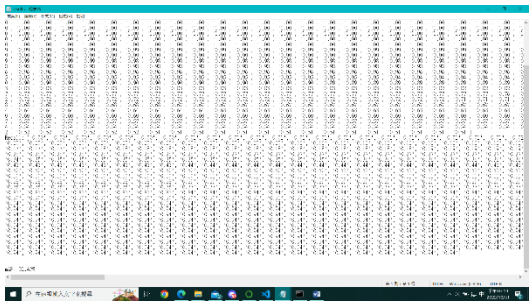
■ Which layer you add SE modules to and compare the corresponding results

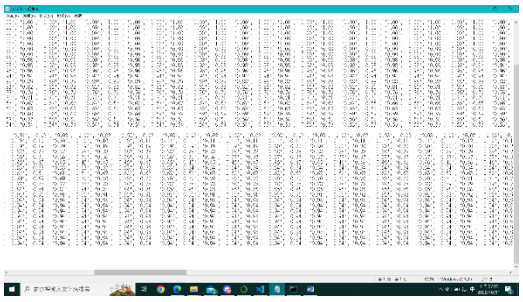
這次加入SE modules主要是對yolo_pafpn.py做修改並加入model裡，主要對backbone部分 dark5 dark4 dark3 output輸出的channel加入SE module，經由下部分會講解的實驗後，得出結論是和完全不加相比，加入SE module反而會使ap微幅上升或下降約1~2%變動，而在一定要加SE module的情況下，選擇在深層dark5 1024 dimension output通道加入3層SE module是最有可能進步的，故這次也是這樣選擇

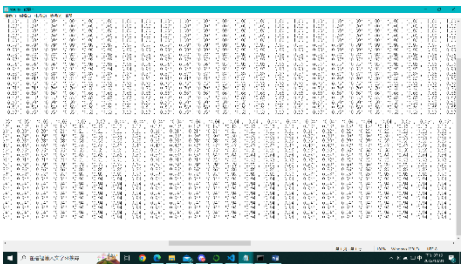
原版未添加以yolo_m跑63 epoch結果(附上map執行結果及recall 完整截圖)

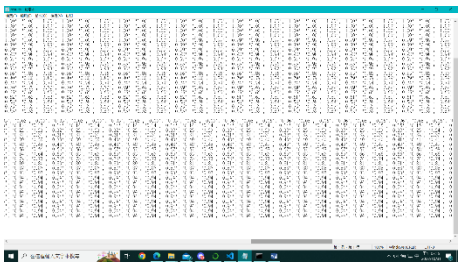


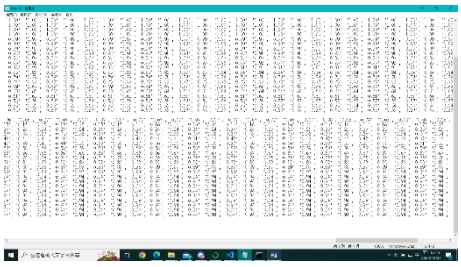


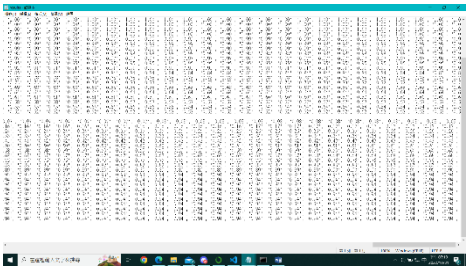






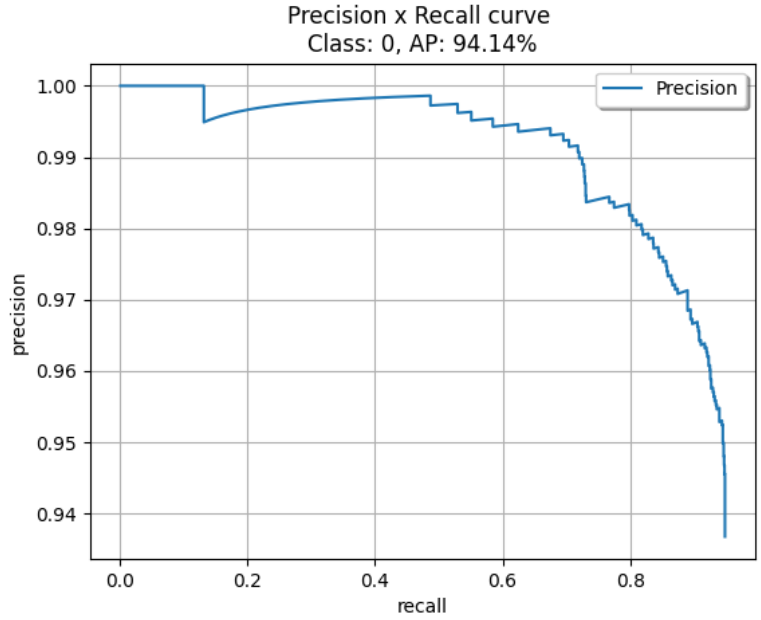


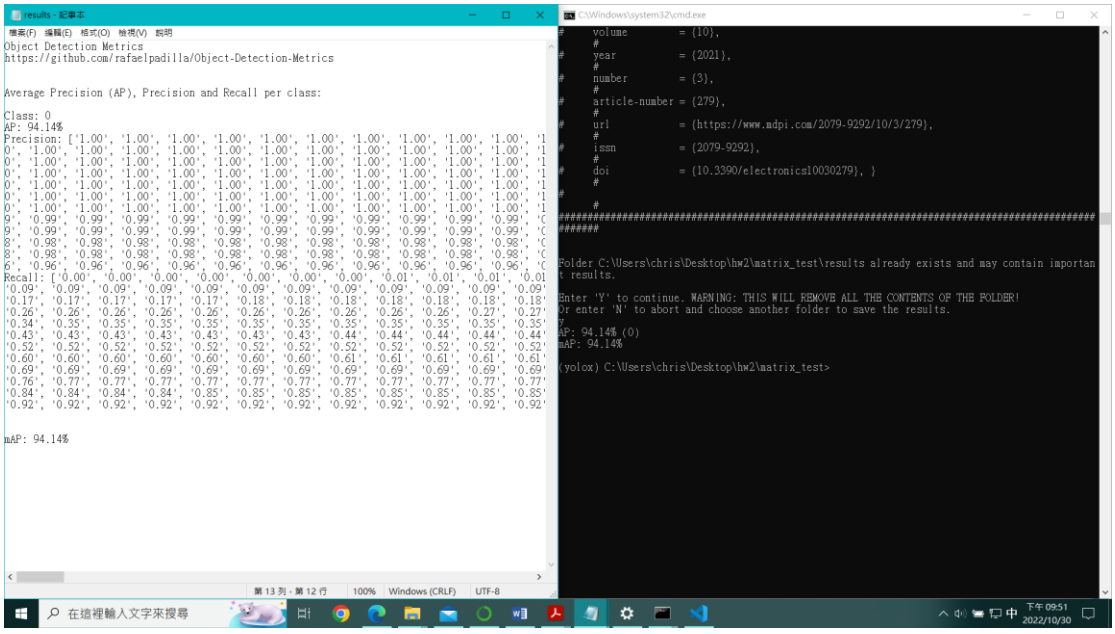


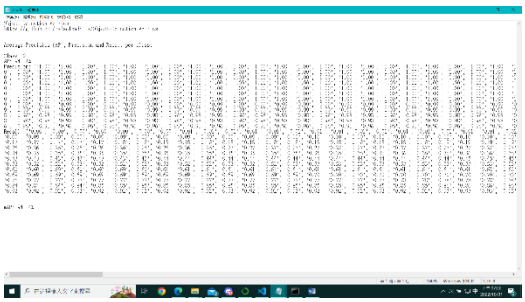


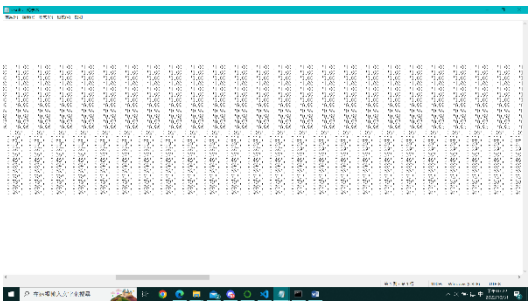
可看出val map是92.43%

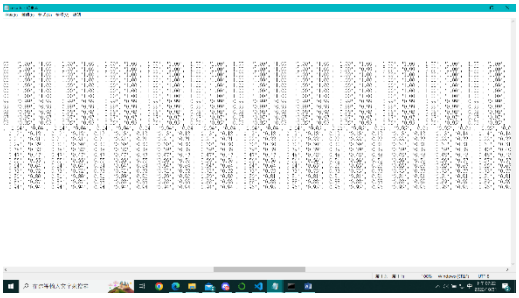
加入se module以yolo_m跑35 epoch後結果如下

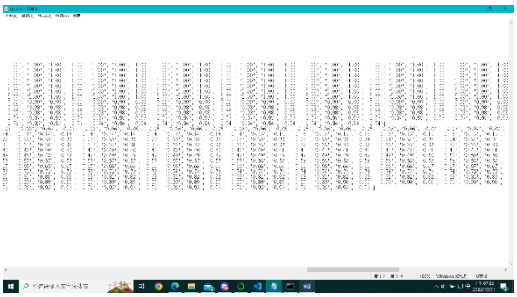

Precision x Recall curve
Class: 0, AP: 94.14%
Precision
precision
recall
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.0
0.2
0.4
0.6
0.8

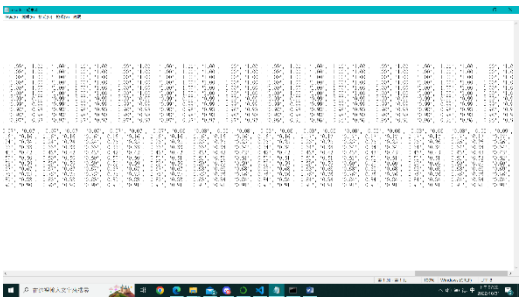

results - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
Object Detection Metrics
https://github.com/rafaelpadilla/Object-Detection-Metrics
Average Precision (AP), Precision and Recall per class:
Class: 0
AP: 94.14%
mAP: 94.14%
C:\Windows\system32\cmd.exe
Folder C:\Users\chris\Desktop\hw2\matrix_test\results already exists and may contain important results.
Enter 'Y' to continue. WARNING: THIS WILL REMOVE ALL THE CONTENTS OF THE FOLDER!
Or enter 'N' to abort and choose another folder to save the results.
AP: 94.14% (0)
mAP: 94.14%
(yolox) C:\Users\chris\Desktop\hw2\matrix_test>











可看出val map是94.19%，代表加入se module後，attention機制確實幫助model有進步，更能檢測出車子

○ (Optional) Analysis

se module測試我是使用yolo_s pretrain model跑epoch 20次看結果(yolox_s.py epoch改成20)

訓練指令如下

python tools/train.py -f exps/example/custom/yolox_s.py -d 0 -b 16 --fp16 -o -c yolox_s.pth

接著用demo.py指令生成val 結果

python tools/demo.py image -f exps/example/custom/yolox_s.py -c YOLOX_outputs/yolox_s/best_ckpt.pth --path assets/val/ --conf 0.25 --nms 0.5 --tsize 640 --save_result --device gpu

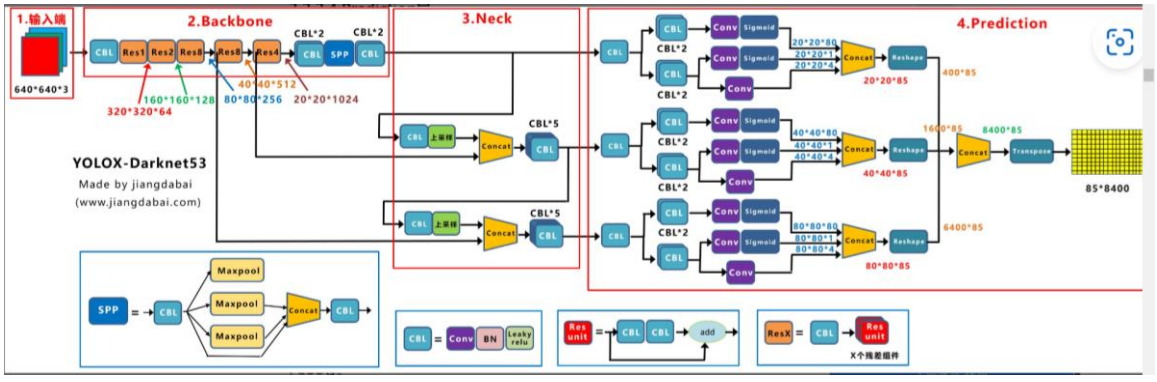
並以本次提供的map計算github跑下列指令

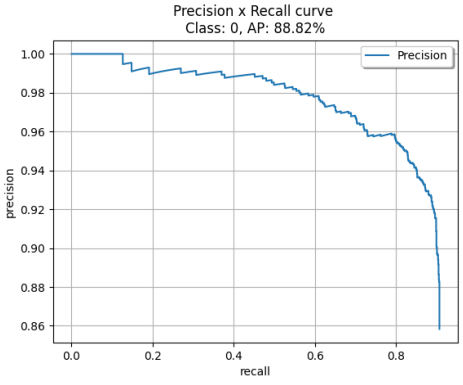
python pascalvoc.py -gt groundtruths_rel/ -gtformat xywh -gtcoords rel -det detections/ -detformat xyrb -imgsize 1920,1080 -t 0.85 -np

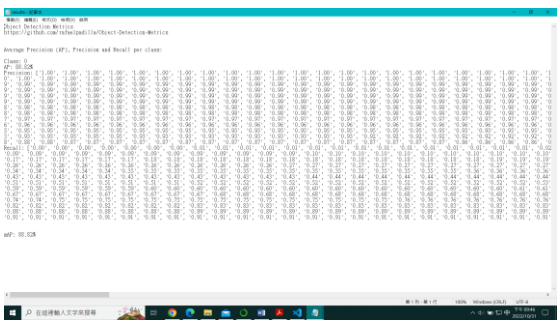
檢測出map數值

未加入SE modules做了兩次測試以防誤差原因是epoch不夠，兩次map分別在88.82 和88.84差距不到0.05

■ The difference between adding to shallow and deep layers

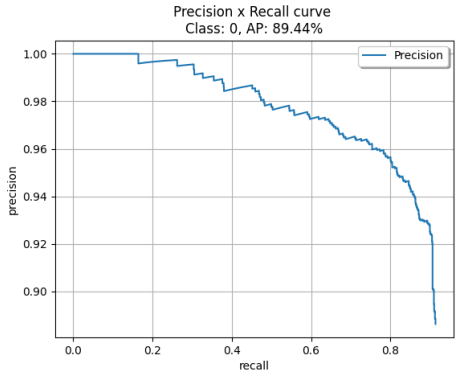


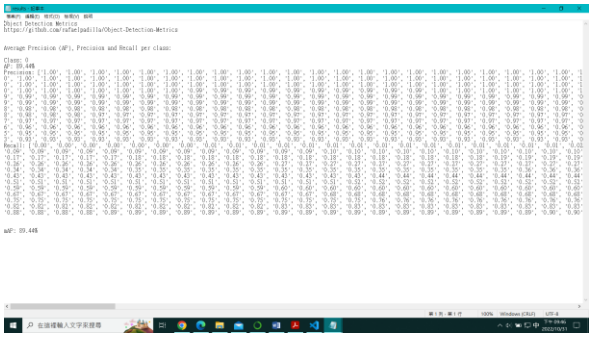




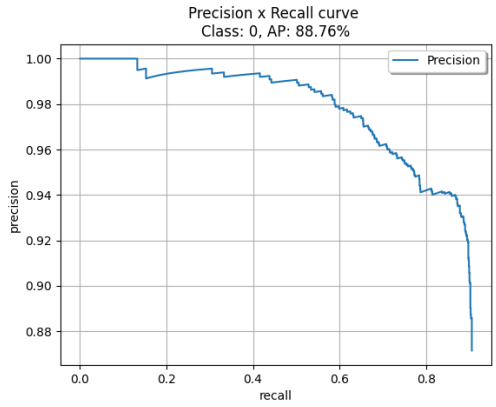
分別在較深層的darknet5和較淺層的darknet3來做實驗看結果

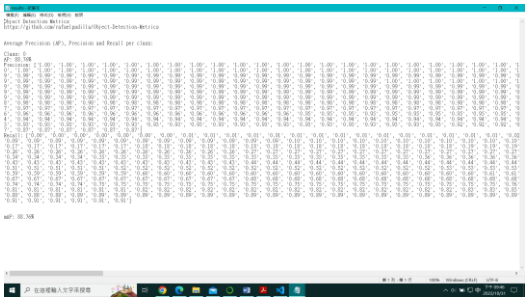
深層部分發現在dark5 1024 dimension output通道加入訓練map結果會是89.44上升約0.6%





淺層部分而在dark3 256 dimension output通道加入訓練map結果會是88.76和原本相比些微下降

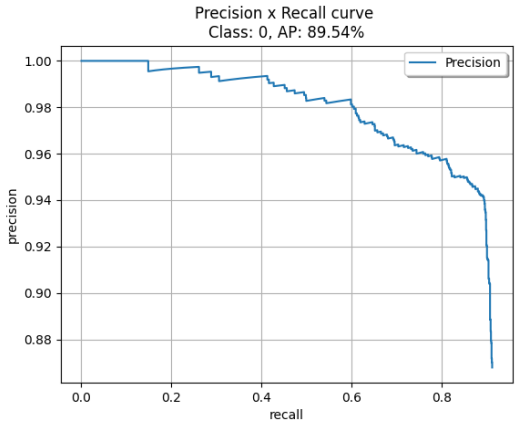


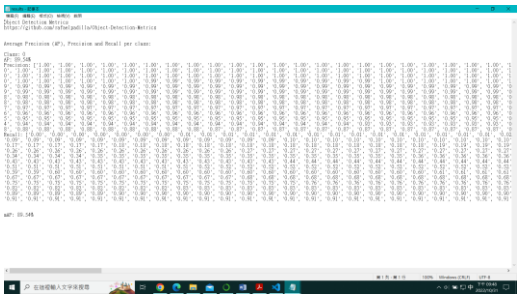


我覺得在深層有進步的原因是attention對壓縮成高維的feature更能一次看到整張圖片，在淺層可能就只能專注在圖片的一部份區塊，單個se module的attetion仍只能專注在一部份區塊，那加入se module可能意義不大

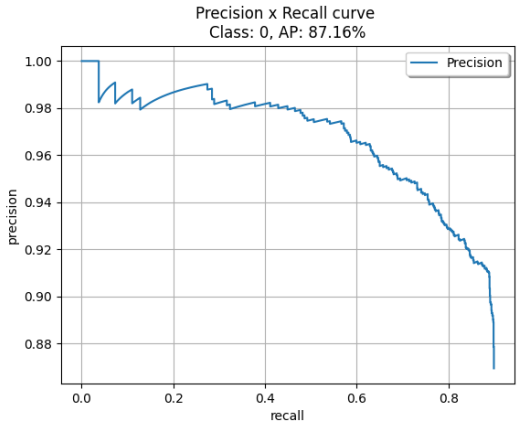
■ The different amount of the SE modules in one layer

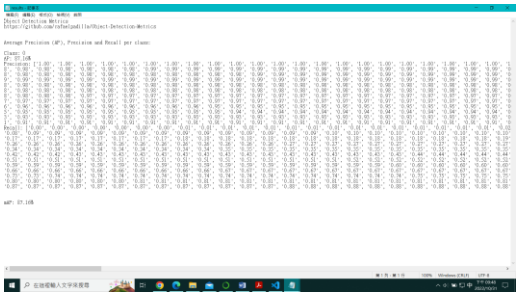
由於上面實驗發現在dark5 1024 dimension output通道加入訓練map結果會是較好的，故這次主要實驗一樣在這層，從原本只加入一個SE module變成三個SE module和五個SE module比較結果，原本1個SE module map結果是89.44，3個SE module map結果會是89.54





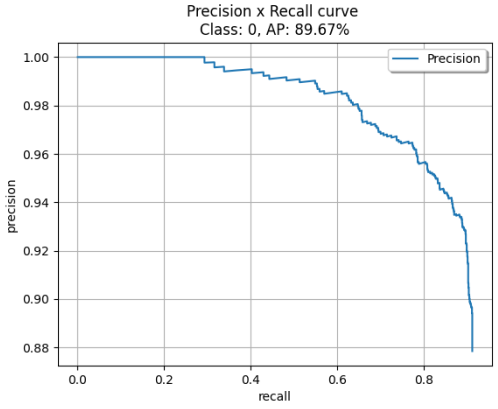
5個SE module map結果會是87.16

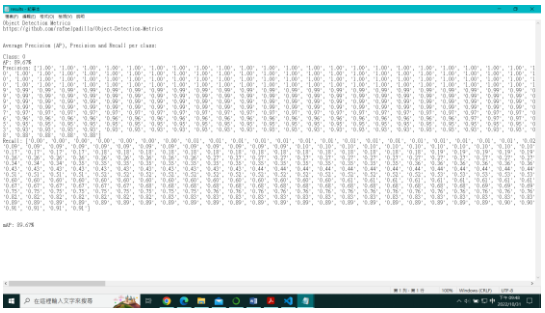




從這可以觀察在同意層出過多的疊加SEmodule並無益處，少量添加有助於更進一步增加效能

此外也有針對dark3 256 dimension output通道設置三個SEmodule試看看結果來確認多層結構對model是否能夠增進效果，原本1個SE module map結果是88.76，3個SE map結果是89.67





意外的是map竟然進步快1%且超過放在dark5 1024全部實驗的成效，我認為這可能是淺層原本只能看到一部份的attention的SE module經過疊加可以看到整張圖的feature，得到在深層加入se一樣的效果

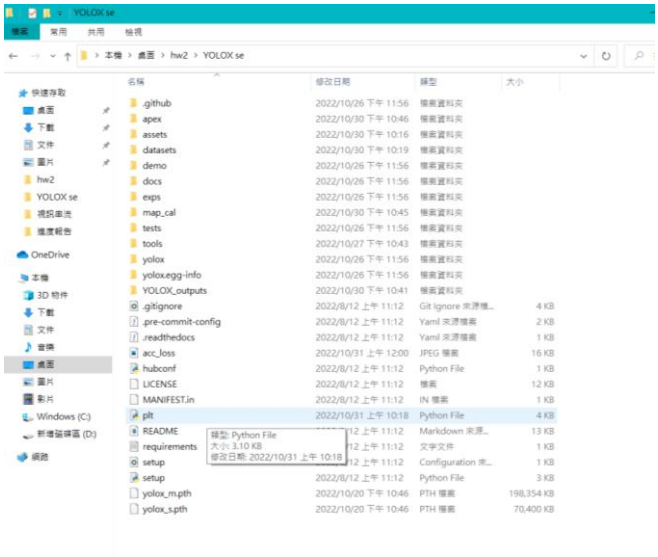
根據上述兩個實驗可以推論出在同一層添加多個SE module有助於增加效果，但僅限於少量(<=3)，添加過多反而會使效能嚴重下降，而本次正式使用se module的是使用在深層dark5 1024放3個SE modu雖然放在淺層dark3 256使用3個SE module在實驗中比較好，但為了確保本次實驗一定是能比原版進步，我選擇在進步看來較穩定的深層dark5 1024放入SE module。

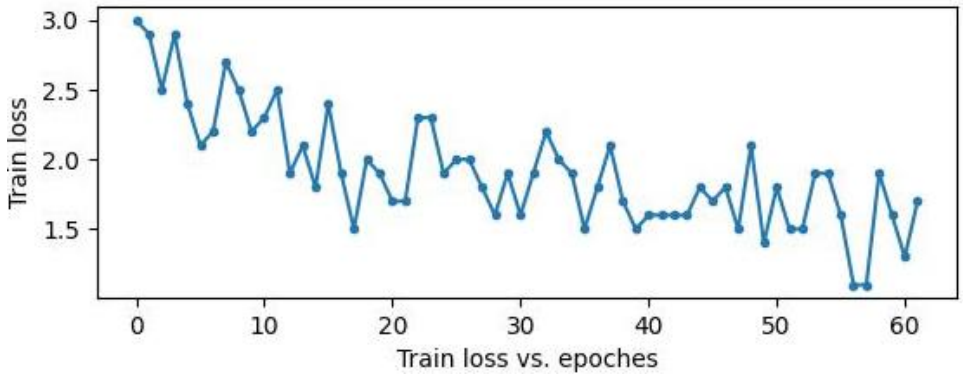
由於詢問助教是否要將這些實驗model weight也放在繳交檔案裡，助教回答可能太大導致檔案無法所以把雲端連結放在這裡，裡面命名方式和下面提到map_cal 裡se_test一樣，連結如下

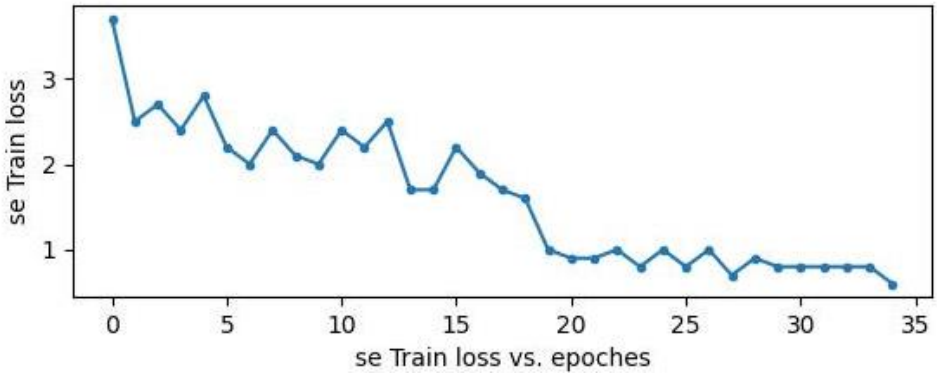
https://drive.google.com/drive/folders/1U35a5G9nnLguezsMvwA42Nhu572TvdVX?usp=share_link

■ Loss training curve, etc.

針對每次 epoch 最後一次 iter 的 total loss(如果把每次 iter 都標出來太過雜亂，故選擇最後一次 iter 做代表看趨勢)當作每次 epoch loss，original epoch 跑 63 次，se epoch 跑 35 次 以下是圖表，我把相關 code 命名 plt.py 放入放在兩個 model 資料夾各自一進去位置如下圖(code loss 是手動輸入並非自動讀 loss 所以出來結果都是兩者檔案附上的 model weight training loss)







Other

Yolo 格式轉換 coco 格式

將相關 code 放在 datasets/hw 裡，參考 reference 將資料排成如下

```
└── $ROOT_PATH
    ├── classes.txt
    ├── images
    └──labels
```

實作將在 hw 資料夾裡放入 train、train_labels、class.txt，train 改名成 images　train_labels 改名成 labels，class.txt 填入本次要分類的類別，本次因只需要偵測 car 故填入 car 即可，之後會輸出一個 annotations 資料夾裡有 train.json ，之後換成 val dataset 做一樣操作(輸出都會是叫 train.json 要自己改名) 並且輸入根目錄$ROOT_PATH 的位置(此處建議輸入目前所在資料夾絕對位置)執行指令

```
python yolo2coco.py --root_dir $ROOT_PATH
```

當初切資料是用

python yolo2coco.py --root_dir

C:\Users\chris\Desktop\hw2\YOLOX\datasets\hw(需改成自己絕對路徑，當時是在桌面的 hw2\yolox 實作的，和繳交檔案的位置不同，並非在 code 寫死，是在 cmd 裡打絕對路徑)

Train/val data 放置地方

將 train/val data 和他們轉換成 coco 格式的 annotations 放在 dataset/final 裡 Train val dataset 改名成和 coco 一樣 train2017 val2017 方便操作

檢測 map 工具

在 map_cal 裡執行

放置 gt 在 map_cal/groundtruth_rel

放置 inference 結果在 map_cal/detections

跑指令

python pascalvoc.py -gt groundtruths_rel/ -gtformat xywh -gtcoords rel -det detections/ -detformat xyrb -imgsize 1920,1080 -t 0.85 -np

-gtformat xywh 代表使用 yolo 格式 -gtcoords 代表使用相對座標 -detformat xyrb 代表使用 coco 格式 -imgsize 1920,1080 給予相對座標之圖片長寬資訊 -np 代表執行結果不直接跳圖表出來

則結果會在 map_cal/results 資料夾顯示出圖表和各檔案數值

修改 output 格式

由於原先 yolox 只有將結果畫到圖片上，沒有保存 txt 結果，所以要做修改 yolox/utils/visualize.py 文件，修改 vis，將 return 資訊改為同時有圖片和 txt(保留圖片方便觀察使用，之後會刪掉圖片部分)

再把 demo.py 裡 call vis 的 function 做修改符合成 vis return 值

最後把 image_demo 裡會把 return image result list 結果改寫成 txt 和圖片 output

在兩份 code 的繳交部分已經把生成圖片的 code 給註解掉了，所以實際執行只會生出本次作業所求的 txt 檔

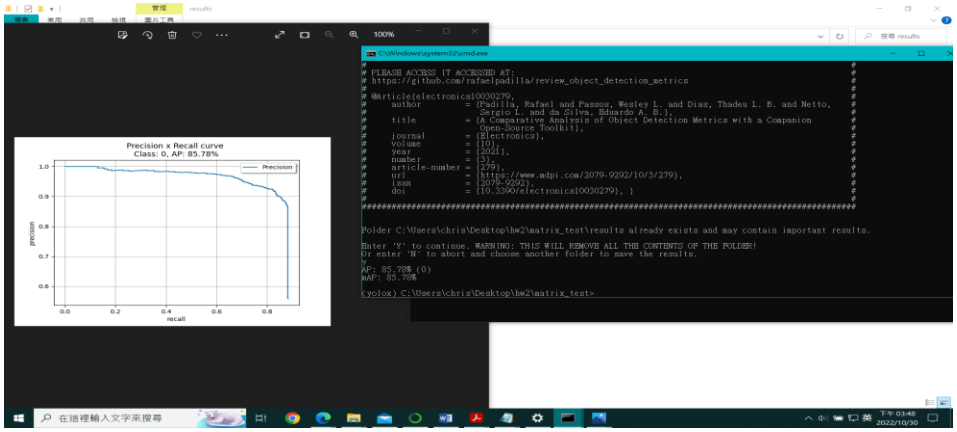
消融實驗

Nms 數值差異

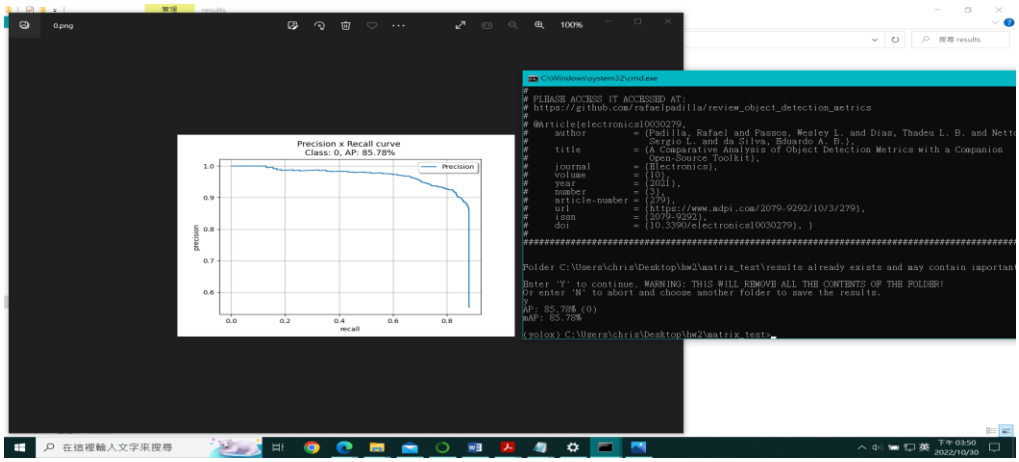
想觀察 demo.py 設置不同 nms 會有不同效能嗎？由於這次 dataset 很多圖片有小的車子重疊在一起，原本是想說 nms 設置低一點會能提升效能，實驗在未添加 se module 用 yolo_x epoch20 跑出的 best checkpoint

但如實驗結果所示完全沒有差異，代表本次 model 在做 nms 前就都能準確框出車子 nms 幾乎沒有刪掉正確的車子，影響不大。

Yolo_s nms0.3



Yolox_s nms0.5



Yolox_s/yolox_m 比較

Yolox_s在跑epoch 35次後，到達map 85.78準確率。Yolox_m則是前面提過的map 92.43，顯示出根據其yolo的weight不同，weight 和 parameter數量越多表現會越好，也就可推測yolox_l成效會比目前我的sota表現會更好，但礙於顯卡效能無法執行大於yolox_m的weight

主要資料路徑

Datasets/final

放有處理好的轉成coco格式label的annotations和train2017(即train資料夾) val2017(即val資料夾)

Datasets/hw

裡面放有yolo2coco.py，根據前面說的將image label class.txt放好執行指令會生成coco格式train.json

Assets

裡面放有demo.py要生成output的inference用圖像繳交檔案有test val兩個空資料夾，分別對應實際繳交test和測量val map數值用的demo檔

Map_cal

計算map，將detections放入inference結果，groundtruths_rel放入要檢測的對應gt，執行報告有提到的指令，同時se_test放有對val做map的各種model result，裡面的origin se256 se256_3 se1024 se1024_3 se1024_5分別對應到前面說過的以yolox_s做標準加入se module實驗結果origin代表沒加，256代表在dark3 output，1024代表在dark5 outpu加入，_3 _5代表同一層加入個數(沒有即代表一個)，final裡有origin val、 se val代表繳交model沒有和有加se的結果。

YOLOX_outputs

訓練時會根據yolo_s.py、yolox_m.py等不同yolo模型將checkpoint存在相對應名字資料夾

YOLOX_outputs/yolo_m

存有本次繳交之model best training weight，/vis_res放有test_result和val_result，分別是本次要inference的test結果和驗證的map val結果

exps/example/custom/yolox_m.py

放有對 yolox_m model 訓練時吃的超參數

yolox/utils/visualize

修改成會讓 output 顯示出 txt 檔，留有原本的未修改檔案

yolox/models/yolo_pafpn.py

本次主要加入 se module 並加以修改處

Tools/demo.py

對輸出格式部分做了修改，讓其可以 output 出 txt 檔，留有未修改的檔案

Tools/eval.py

有試著做修改但沒使用，留有未修改的檔案

Original/SE

分別放置這次有無添加 SE module 之最好結果生成的 test txt 結果

Yolox_m.pth

從 yolox github 下載的 pretrained weight 放在和 yolox github clone 下來同個資料夾，在 train.py 時會使用

主要使用指令

Inference

在 Original/SE 資料夾裡

python tools/demo.py image -f exps/example/custom/yolox_m.py -c YOLOX_outputs/yolox_m/best_ckpt.pth --path assets/test/ --conf 0.25 --nms 0.5 --tsize 640 --save_result --device gpu

train

在 Original/SE 資料夾裡

python tools/train.py -f exps/example/custom/yolox_m.py -d 0 -b 8 --fp16 -o -c yolox_m.pth

map 計算

進入 map_cal 裡

python pascalvoc.py -gt groundtruths_rel/ -gtformat xywh -gtcoords rel -det detections/ -detformat xyrb -imgsize 1920,1080 -t 0.85 -np