計算機視覺

Computer Vision

Homework IV

國立清華大學

系所:電子所碩二

中文姓名:李聖謙

學號:111063517

授課老師:Min Sun, 孫民

Contents

Discussion graph meaning

I. Confusion matrix

1. TP ： 正例被預測成正例(預測正確正例)
2. FP ： 反例被預測成正例(預測錯誤正例)
3. FN ： 反例被預測成正例(預測錯誤反例)
4. TP ： 反例被預測成反例(預測正確反例)

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}} = \frac{\text{預測正確正例}}{\text{預測正確正例} + \text{預測錯物正例}}$$

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} = \frac{\text{預測正確正例}}{\text{預測正確正例} + \text{預測錯誤反例}}$$

由 Precision 的公式可知，Precision 是站在預測結果的角度進行的計算，也就是想看正例預測結果中正確的比例，而 Recall 是站在事實角度進行的計算，也就是想看實際正例有多少被正確預測。

II. PR curve

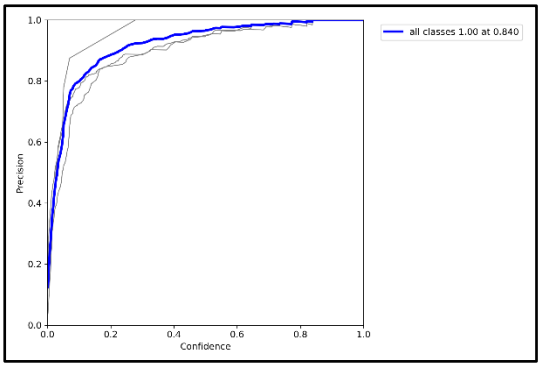


Figure 1 P curve

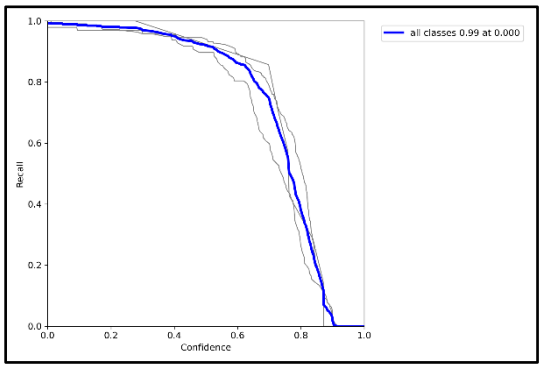


Figure 2 R curve

當 confidence 越高時，表示 model 只有在非常確信時才會預測為正例，如此會讓預測的結果中 precision 很高，因為預測的嚴謹度提高了，所以 FP 值會下降，但這其實只是在刻意提高 precision，因為這個 model 在提升 precision 時也漏掉了很多實際該辨別為正例的照片沒被辨別，FN 的值上升因此 recall 會大幅降低。

相反的，當 confidence 越低時，表示 model 只要在目標稍微有可能是正例時就會預測為正例，因嚴謹度下降，故實際上的正例大多都會被預測到，但同時也會捕捉到更多反例(FP 上升)，所以 precision 降低，recall 上升。

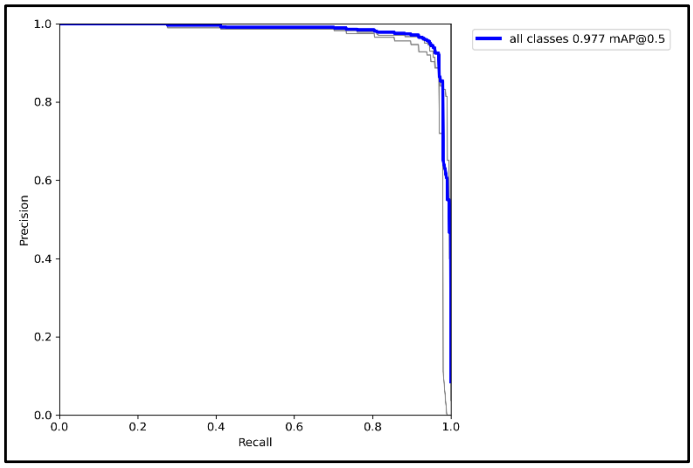


Figure 3 PR curve

mAP 的是意思是 PR curve 底下的面積，在理想的情形下，precision 和 recall 值都是接近 1，所以 mAP 的理想值應該為 1，但實際上 P 跟 R 會如同上面討論會有 trade off，透過 PR curve 可以讓我們知道這個 model 的在特定的 confidence 下的 performance，面積越大代表是一個好的 model。

本次題目的 mAP 有討論了兩個分別是@0.5 和@0.5:0.95，從結果上來說前者的 mAP 都較後者來的大，我推測是因為後者是取 0.5 到 0.95 之間好幾個 confidence 的 mAP 做平均，而隨著 confidence 上升，mAP 下降，所以@0.5 的結果才會比@0.5:0.95 佳。

III. F1 curve

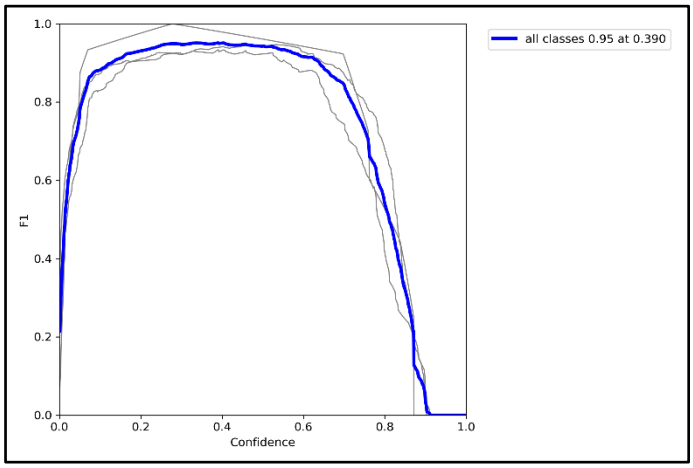


Figure 4 F1 curve

$$\boldsymbol{F1\ score} = (\mathbf{1}+\boldsymbol{\beta^2})\frac{\boldsymbol{P*R}}{\boldsymbol{\beta^2(P+R)}} \rightarrow \boldsymbol{\beta}=\mathbf{1} \rightarrow = \frac{\mathbf{2}*\boldsymbol{P}*\boldsymbol{R}}{\boldsymbol{(P+R)}}$$

F1 curve 的是 Precision 和 Recall 的調和平均數，越接近 1 表示 model 在 Precision 和 Recall 之間取得了更好的平衡，有助於找出合適的 confidence，但這並不是高就是好的，最終還是取決於 model 的應用需要高 Precision 或是高 Recall。

Q1

Find the split train val path function in data/preprocess.py and write the code to generate the training YAML file using the Q1 dataset. (20%)

I. Version1 (train_val_ratio=0.9)

Version1 是未分類時的結果，因為未分類，所以照片 split 後會按照順序，train dataset 的照片都是 398 資料夾(共 160 張，全部進 train)的照片，而 validation 的資料夾因為較少，所以只包含了 173 資料夾(共 40 張，前 20 張進 train，後 20 張進 validation)的照片。

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     11.7G   0.03176   0.03653  0.003146   0.07143       488       640: 100% 15/15 [00:13<00:00,  1.08it/s]
          Class    Images    Labels         P         R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.35it/s]
            all        20       147     0.687     0.831     0.798     0.418

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     11.7G   0.03274   0.04319   0.00378   0.07971       373       640: 100% 15/15 [00:14<00:00,  1.04it/s]
          Class    Images    Labels         P         R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.87it/s]
            all        20       147     0.716     0.851     0.794     0.403

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     11.7G   0.02794   0.03781  0.003294   0.06904       583       640: 100% 15/15 [00:14<00:00,  1.06it/s]
          Class    Images    Labels         P         R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.10it/s]
            all        20       147      0.66     0.847     0.791     0.419
```

Figure 5 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.523	0.437	0.46	0.287
	car	1200	1504	0.795	0.608	0.743	0.425
	bus	1200	210	0.704	0.362	0.527	0.36
	truck	1200	47	0.07	0.34	0.11	0.076
495							
	all	1200	2227	0.663	0.626	0.615	0.367
	car	1200	1950	0.696	0.621	0.663	0.374
	bus	1200	55	0.911	0.636	0.732	0.502
	truck	1200	222	0.383	0.622	0.449	0.226
410							
	all	1200	2331	0.569	0.554	0.581	0.376
	car	1200	2005	0.875	0.566	0.72	0.423
	bus	1200	15	0.209	0.533	0.418	0.355
	truck	1200	311	0.625	0.563	0.605	0.35
511							
	all	1200	2294	0.523	0.407	0.412	0.255
	car	1200	2077	0.877	0.403	0.684	0.384
	bus	1200	86	0.509	0.314	0.32	0.217
	truck	1200	131	0.183	0.504	0.234	0.166
398							
	all	1200	2353	0.742	0.665	0.684	0.448
	car	1200	2056	0.74	0.737	0.738	0.433
	bus	1200	104	0.912	0.501	0.669	0.478
	truck	1200	193	0.574	0.756	0.645	0.432
173							
	all	1200	1991	0.844	0.634	0.788	0.499
	car	1200	1680	0.901	0.647	0.846	0.531
	bus	1200	171	0.885	0.754	0.875	0.6
	truck	1200	140	0.746	0.5	0.642	0.365
ALL							
	all	1200	12957	0.666	0.55	0.579	0.36
	car	1200	11272	0.813	0.58	0.711	0.414
	bus	1200	641	0.766	0.466	0.572	0.397
	truck	1200	1044	0.419	0.603	0.453	0.269

Figure 6 Test results

II. **Version2** (train_val_ratio=0.9)

Version2 的分類方式因為只有 2 個資料夾 200 張照片，所以我選擇一樣在 train_val_ratio=0.9 的情況下，但 train 和 validation 的 dataset 都必須包含 2 個資料夾的 data，我採取的方式如下。

```python
def split_train_val_path(all_image_paths, train_val_ratio=0.9):

    camera_398_paths = all_image_paths[:160]
    random.shuffle(camera_398_paths)

    camera_173_paths = all_image_paths[160:]
    random.shuffle(camera_173_paths)

    train_image_paths = camera_398_paths[: int(len(camera_398_paths) * train_val_ratio)] + camera_173_paths[: int(len(camera_173_paths) * train_val_ratio)]
    val_image_paths = camera_398_paths[int(len(camera_398_paths) * train_val_ratio):] + camera_173_paths[int(len(camera_173_paths) * train_val_ratio):]

    return train_image_paths, val_image_paths
```

Figure 7 Q1_ver2 code

因為依照 all_image_path 讀取 dataset 的路徑會先寫入 398 資料夾的路徑，再寫入 173 資料夾的路徑，所以我將 all_image_path 取前 160 張存成 camera_398_paths，後 40 張存成 camera_173_paths，再將 camera_398_paths 和 camera_173_paths 內的路徑做隨機處理，最後按照，train_val_ratio=0.9 分配到 train_image_path 和 val_image_path，如此確定 train 和 validation 資料夾內皆含有 2 個資料夾的 data。

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     11.3G   0.02745   0.03767  0.003589    0.0687       522       640: 100% 15/15 [00:14<00:00,   1.06it/s]
           Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,   1.64it/s]
             all          20         325       0.948       0.955       0.977       0.739
```

Figure 8 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.517	0.482	0.476	0.292
	car	1200	1504	0.856	0.61	0.758	0.415
	bus	1200	210	0.642	0.452	0.57	0.399
	truck	1200	47	0.052	0.383	0.1	0.062
495							
	all	1200	2227	0.664	0.61	0.632	0.385
	car	1200	1950	0.727	0.619	0.669	0.366
	bus	1200	55	0.853	0.632	0.787	0.564
	truck	1200	222	0.413	0.581	0.44	0.224
410							
	all	1200	2331	0.535	0.593	0.512	0.296
	car	1200	2005	0.862	0.592	0.736	0.422
	bus	1200	15	0.151	0.533	0.201	0.166
	truck	1200	311	0.591	0.653	0.599	0.3
511							
	all	1200	2294	0.569	0.455	0.466	0.287
	car	1200	2077	0.929	0.388	0.698	0.371
	bus	1200	86	0.583	0.488	0.456	0.319
	truck	1200	131	0.195	0.489	0.244	0.171
398							
	all	1200	2353	0.779	0.642	0.709	0.484
	car	1200	2056	0.77	0.675	0.751	0.437
	bus	1200	104	0.931	0.516	0.676	0.514
	truck	1200	193	0.637	0.736	0.702	0.5
173							
	all	1200	1991	0.812	0.75	0.806	0.514
	car	1200	1680	0.939	0.711	0.891	0.545
	bus	1200	171	0.844	0.895	0.865	0.615
	truck	1200	140	0.653	0.643	0.663	0.381
ALL							
	all	1200	12957	0.673	0.569	0.601	0.372
	car	1200	11272	0.831	0.572	0.723	0.409
	bus	1200	641	0.743	0.524	0.597	0.428
	truck	1200	1044	0.444	0.61	0.484	0.28

Figure 9 Test result

Compare

i. Train PR curve

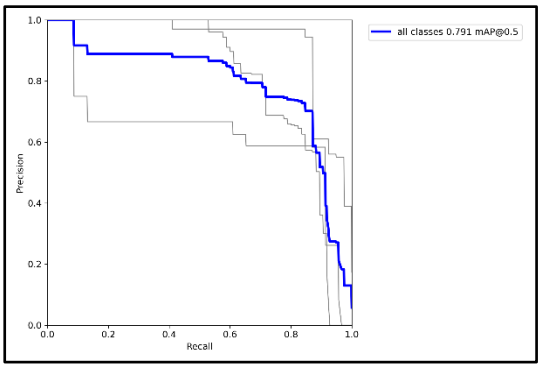


Figure 10 Ver1

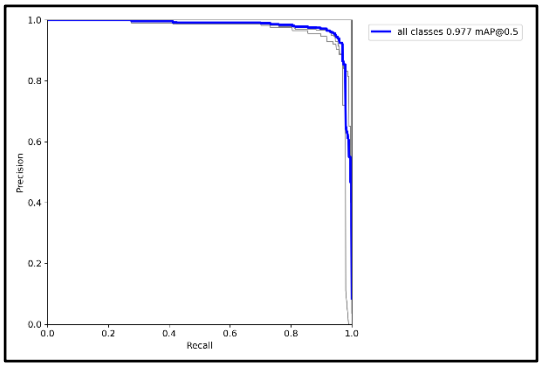


Figure 11 Ver2

ii. Test PR curve

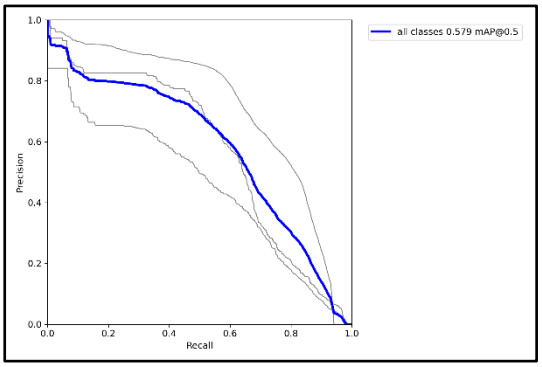


Figure 12 Ver1

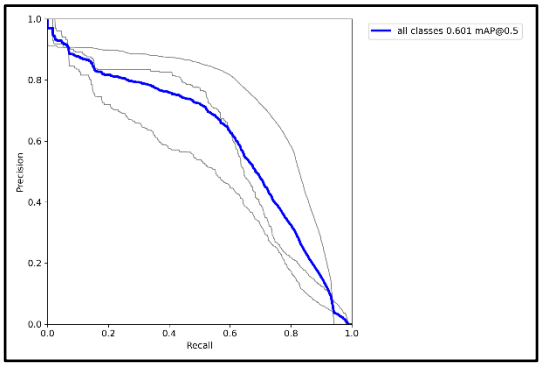


Figure 13 Ver2

從結果可以發現 mAP 的的值因為不同的 split 方法，而在 test 上有些微的上升，而在 train 的 PR curve，已經接近理想值 mAP = 1，所以這個分類資料夾的方法是可以有效提升 training model performance，原因我認為是因為在 train 的過程中 validation 的資料先看過了兩個資料夾的 data，可以比起只看一個資料夾時有更精確的預測；但 test 的 mAP 還不高的原因是因為 data 的泛化能力太低，只訓練了 173 和 398 的 data 但測試的還包含額外的 4 個資料夾，從 test 的結果可以發現沒被訓練的 data 表現的 mAP 都較有訓練過的 data 值低，有訓練的結果都是至少 0.7 以上，但未訓練的最高只有 0.6 最低只有 0.4 左右，故整體的 mAP 被為訓練的 data 拉低了平均。

iii. Train F1 curve

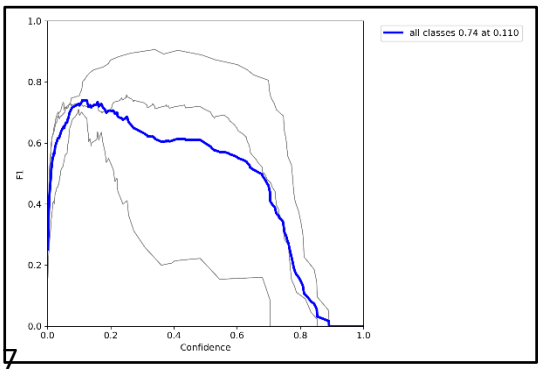


Figure 14 Ver1

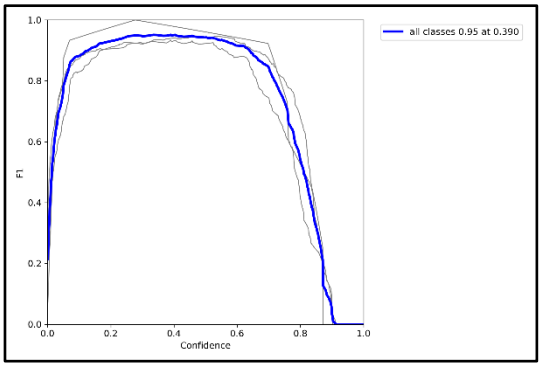


Figure 15 Ver2

iv. Test F1 curve

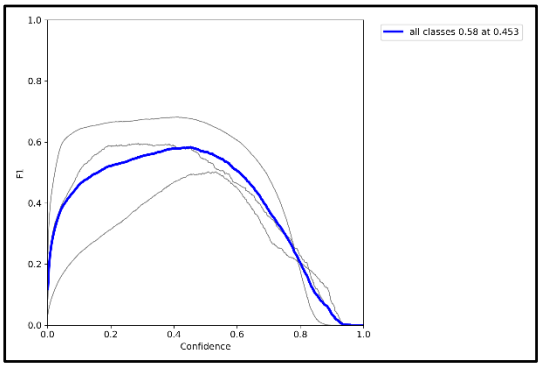


Figure 17 Ver1

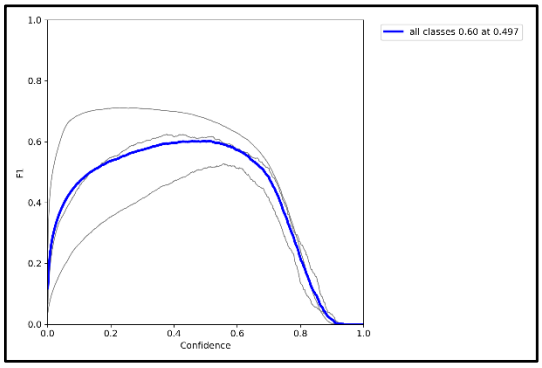


Figure 16 Ver2

在 train 的 F1 curve 可以發現 ver2 的 F1 curve 比 ver1 好上很多，我認為是因為經過資料夾分類，對 train 的 precision 和 recall 都有提升；test 的 F1 curve 上，ver1 和 ver2 似乎沒什麼差別，我推測可能是因為訓練的資料太少，所以 test 的結果都不是很理想，不管有沒有對照片分類都無法再將 F1 提高，雖然 test 都不甚理想，但我們至少可以得到結論是 ver2 是一個較佳的訓練 model。

Q2

Propose a solution to sample the same amount of data as the Q1 dataset from the Q2 dataset, which has more data, and finish the select images function in data/preprocess.py to generate the training YAML file. (35%)

本小題的目的是透過訓練更多不同的 DATA 已取得更佳的泛化能力。

I. **Version1** (train_val_ratio=0.9)

因為不確定每個資料夾對 model 的影響力，所以我第一個選擇 data 的方法採取平均選擇的策略，6 個資料夾各選至少 30 張當作 training data，方法如下。

```python
def select_imaegs(image_paths, images_num=200):
    #ver1

    num_folders = 6
    num_images_per_folder = 200
    total_images = num_folders * num_images_per_folder

    # 先隨機取每個資料夾中的30張
    selected_image_paths = []

    for folder_start in range(0, total_images, num_images_per_folder):
        folder_paths = image_paths[folder_start : folder_start + num_images_per_folder]
        random.shuffle(folder_paths)
        selected_image_paths.extend(folder_paths[:30])

    remaining_images = list(set(image_paths) - set(selected_image_paths))
    remaining_images_number = len(remaining_images)

    for folder_start in range(0, remaining_images_number, 180):
        folder_paths = remaining_images[folder_start : folder_start + 180]
        random.shuffle(folder_paths)
        selected_image_paths.extend(folder_paths[30:33])

    # 再隨機取剩下的未被選上的20張
    selct_2 = list(set(image_paths) - set(selected_image_paths))
    random.shuffle(selct_2)
    selected_image_paths.extend(selct_2[:2])

    return selected_image_paths
```

Figure 18 Select image code

這部分的照片選擇則邏輯是，因為 image_path 在引入時會按照資料夾排序，所以每 200 張照片就會換一張，第一個 for 迴圈，在一次迴圈內會從 200 張中先進行隨機亂數處理後，選出前 30 張存進 selcet_image_paths，共 6 次迴圈，得到 180 條 paths，並將未被選上的 paths 存進 remaining_images 中，剩餘的 20 張再從 remaining_images 隨機挑選存進 selcet_image_paths。

```python
def split_train_val_path(all_image_paths, train_val_ratio=0.9):
    # ver1

    random.shuffle(all_image_paths)


    train_image_paths = all_image_paths[: int(len(all_image_paths) * train_val_ratio)]  # just an example
    val_image_paths = all_image_paths[int(len(all_image_paths) * train_val_ratio):]  # just an example

    return train_image_paths, val_image_paths
```

Figure 19 Split code

隨機分配路徑到 train_image_paths 和 val_image_path。

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     11.1G   0.03286   0.01862  0.003789   0.05527       239       640: 100% 15/15 [00:13<00:00,  1.14it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.56it/s]
             all         20        157      0.929      0.803        0.9      0.647

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     11.1G   0.03064   0.02197  0.004177   0.05679       125       640: 100% 15/15 [00:13<00:00,  1.09it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.25it/s]
             all         20        157      0.911      0.782      0.897      0.566

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     11.1G   0.03032   0.01942   0.00396    0.0537       238       640: 100% 15/15 [00:12<00:00,  1.16it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.41it/s]
             all         20        157      0.893      0.806      0.906      0.663
```

Figure 20 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.633	0.651	0.643	0.419
	car	1200	1504	0.923	0.779	0.877	0.513
	bus	1200	210	0.713	0.814	0.82	0.572
	truck	1200	47	0.261	0.361	0.233	0.173
495							
	all	1200	2227	0.721	0.734	0.735	0.448
	car	1200	1950	0.686	0.87	0.775	0.458
	bus	1200	55	0.86	0.836	0.864	0.55
	truck	1200	222	0.618	0.495	0.566	0.335
410							
	all	1200	2331	0.623	0.827	0.718	0.489
	car	1200	2005	0.857	0.774	0.819	0.504
	bus	1200	15	0.193	0.863	0.468	0.401
	truck	1200	311	0.818	0.842	0.867	0.562
511							
	all	1200	2294	0.768	0.612	0.654	0.424
	car	1200	2077	0.895	0.812	0.896	0.52
	bus	1200	86	0.782	0.628	0.653	0.473
	truck	1200	131	0.626	0.397	0.413	0.278
398							
	all	1200	2353	0.763	0.705	0.743	0.497
	car	1200	2056	0.736	0.791	0.773	0.473
	bus	1200	104	0.915	0.625	0.792	0.581
	truck	1200	193	0.638	0.699	0.665	0.437
173							
	all	1200	1991	0.82	0.797	0.86	0.574
	car	1200	1680	0.911	0.838	0.924	0.601
	bus	1200	171	0.821	0.859	0.866	0.614
	truck	1200	140	0.729	0.693	0.79	0.509
ALL							
	all	1200	12957	0.742	0.732	0.75	0.487
	car	1200	11272	0.805	0.802	0.82	0.496
	bus	1200	641	0.752	0.758	0.769	0.542
	truck	1200	1044	0.669	0.635	0.661	0.422

Figure 21 Test result

以結果來說 test 的 mAP 是不錯的，這個 select 方法有一個缺點是不確定因素太大，可能會再某個資料夾選太多，或太少導致 model 的表現上下限差異很大，是一個不太穩定的 model，但經過好幾次 training 的結果顯示 mAP 至少會維持在 0.7 左右，穩定的比 Q1 的 performance 來的好。

II. **Version2** (train_val_ratio=0.9)

為了彌補 version1 的不穩定性，我想另了一個 select 方法，不只可以確保 data 的數量不會太懸殊，還可以透過 version1 的結果知道在不同資料夾對 mAP 的影響力，來調整資料夾的 data 數量，此處的策略是將 version1 中訓練較差的資料夾，增加取出照片的數量，透過訓練更多的 data 以達到更高的 mAP。

```python
def select_imaegs(image_paths, images_num=200):
    #ver2

    selected_image_paths = []

    camera_173_paths = image_paths[:200]
    random.shuffle(camera_173_paths)

    camera_398_paths = image_paths[200:400]
    random.shuffle(camera_398_paths)

    camera_170_paths = image_paths[400:600]
    random.shuffle(camera_170_paths)

    camera_410_paths = image_paths[600:800]
    random.shuffle(camera_410_paths)

    camera_511_paths = image_paths[800:1000]
    random.shuffle(camera_511_paths)

    camera_495_paths = image_paths[1000:]
    random.shuffle(camera_495_paths)

    selected_image_paths.extend(camera_173_paths[:20])
    selected_image_paths.extend(camera_398_paths[:35])
    selected_image_paths.extend(camera_170_paths[:45])
    selected_image_paths.extend(camera_410_paths[:45])
    selected_image_paths.extend(camera_511_paths[:20])
    selected_image_paths.extend(camera_495_paths[:35])

    return selected_image_paths
```

區分 6 個資料夾路徑，並隨機處理

可自行控制每個資料夾資料選取量

Figure 22 Select image code

從 version1 的結果可以發現，170 和 410 的 mAP 都較低，所以我選擇最多照片訓練，398 和 495 是中間值，故選擇次多的照片量，173 和 511 只要用少量的照片就可以獲得不錯的訓練效果，採取的方法是先將 image_path 的照片按照原來的資料夾分成 6 個不同的路徑，再隨機亂數後選擇各自路徑的前幾張照片存進 selcet_image_path，如此可以確保每個資料夾都取道我想要的數量。

```python
def split_train_val_path(all_image_paths, train_val_ratio=0.9):
    # ver2
    import re
    folders = {'173': [], '398': [], '170': [], '410': [], '511': [], '495': []}
    # 提取文件路徑中的數字並進行分類
    for path in all_image_paths:
        match = re.search(r'/(\d+)-', path)
        if match:
            folder_number = match.group(1)
            if folder_number in folders:
                folders[folder_number].append(path)

    random.shuffle(folders['173'])
    camera_173_paths_train = folders['173'][: int(len(folders['173']) * train_val_ratio)]
    camera_173_paths_val = folders['173'][int(len(folders['173']) * train_val_ratio):]

    random.shuffle(folders['398'])
    camera_398_paths_train = folders['398'][: int(len(folders['398']) * train_val_ratio)]
    camera_398_paths_val = folders['398'][int(len(folders['398']) * train_val_ratio):]

    random.shuffle(folders['170'])
    camera_170_paths_train = folders['170'][: int(len(folders['170']) * train_val_ratio)]
    camera_170_paths_val = folders['170'][int(len(folders['170']) * train_val_ratio):]

    random.shuffle(folders['410'])
    camera_410_paths_train = folders['410'][: int(len(folders['410']) * train_val_ratio)]
    camera_410_paths_val = folders['410'][int(len(folders['410']) * train_val_ratio):]

    random.shuffle(folders['511'])
    camera_511_paths_train = folders['511'][: int(len(folders['511']) * train_val_ratio)]
    camera_511_paths_val = folders['511'][int(len(folders['511']) * train_val_ratio):]

    random.shuffle(folders['495'])
    camera_495_paths_train = folders['495'][: int(len(folders['495']) * train_val_ratio)]
    camera_495_paths_val = folders['495'][int(len(folders['495']) * train_val_ratio):]

    train_image_paths = camera_173_paths_train + camera_398_paths_train + camera_170_paths_train + camera_410_paths_train + camera_511_paths_train + camera_495_paths_train
    val_image_paths = camera_173_paths_val + camera_398_paths_val + camera_170_paths_val + camera_410_paths_val + camera_511_paths_val + camera_495_paths_val

    return train_image_paths, val_image_paths
```

區分 6 個資料夾路徑，並存成各自的名稱(folders)。

將選取的照片依照 ratio 分成 train 和 validation

Figure 23 Split code

讀取 all_image_paths 並在 for 迴圈內偵測 path 的路徑上的資料夾數字存入入先準備好的陣列裡，再將分好的資料夾作亂數分成 train_image_path 和 val_image_path。

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     10.5G   0.02941   0.02159   0.00301   0.05401       139       640: 100% 15/15 [00:12<00:00,   1.22it/s]
           Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,   2.94it/s]
             all         22        173       0.92      0.904       0.94      0.634

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     10.5G   0.02863   0.02232  0.003129   0.05409       158       640: 100% 15/15 [00:11<00:00,   1.30it/s]
           Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,   2.46it/s]
             all         22        173      0.932      0.909      0.953      0.657

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     10.5G   0.02929   0.02423  0.003262   0.05678       212       640: 100% 15/15 [00:13<00:00,   1.11it/s]
           Class     Images     Labels          P          R     mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,   1.31it/s]
             all         22        173      0.928      0.916      0.958      0.673
```

Figure 24 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.573	0.686	0.653	0.396
	car	1200	1504	0.834	0.844	0.888	0.498
	bus	1200	210	0.707	0.852	0.862	0.56
	truck	1200	47	0.179	0.361	0.207	0.131
495							
	all	1200	2227	0.69	0.798	0.762	0.454
	car	1200	1950	0.687	0.888	0.799	0.43
	bus	1200	55	0.862	0.8	0.867	0.615
	truck	1200	222	0.519	0.707	0.62	0.318
410							
	all	1200	2331	0.696	0.72	0.756	0.462
	car	1200	2005	0.857	0.734	0.818	0.469
	bus	1200	15	0.363	0.8	0.611	0.427
	truck	1200	311	0.867	0.627	0.838	0.489
511							
	all	1200	2294	0.78	0.7	0.727	0.471
	car	1200	2077	0.852	0.782	0.86	0.461
	bus	1200	86	0.869	0.884	0.885	0.669
	truck	1200	131	0.619	0.435	0.436	0.284
398							
	all	1200	2353	0.753	0.682	0.768	0.488
	car	1200	2056	0.733	0.784	0.783	0.442
	bus	1200	104	0.9	0.521	0.813	0.56
	truck	1200	193	0.627	0.741	0.708	0.461
173							
	all	1200	1991	0.799	0.689	0.793	0.502
	car	1200	1680	0.87	0.87	0.918	0.562
	bus	1200	171	0.849	0.625	0.8	0.514
	truck	1200	140	0.677	0.571	0.662	0.432
ALL							
	all	1200	12957	0.728	0.725	0.764	0.465
	car	1200	11272	0.788	0.82	0.833	0.468
	bus	1200	641	0.772	0.691	0.802	0.537
	truck	1200	1044	0.626	0.664	0.657	0.391

Figure 25 Test result

從結果可以發現現在每個資料夾的 mAP 都已經變的平均，雖然 170 略低了一點，但整體還是都在 0.7 以上，所以解決了 ver1 的不穩定性，這個選擇照片的方法，可以維持較穩定 mAP。

Compare

i. Train PR curve

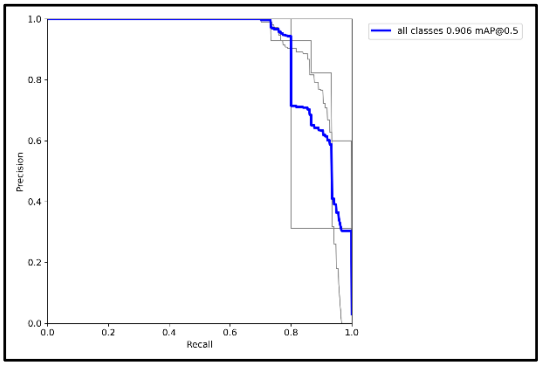


Figure 26 Ver1

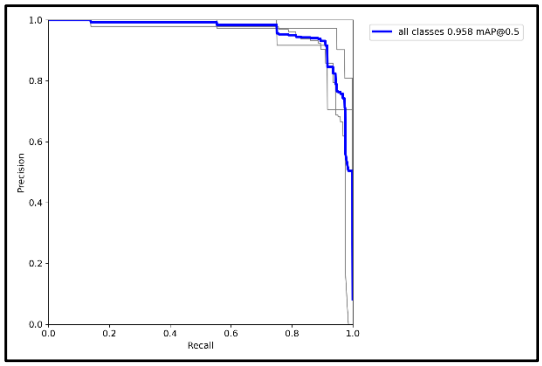


Figure 27 Ver2

ii. Test PR curve

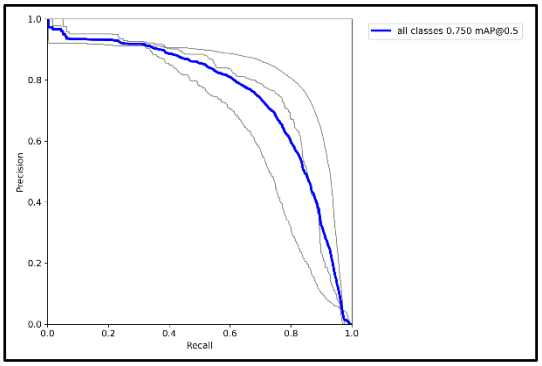


Figure 28 Ver1

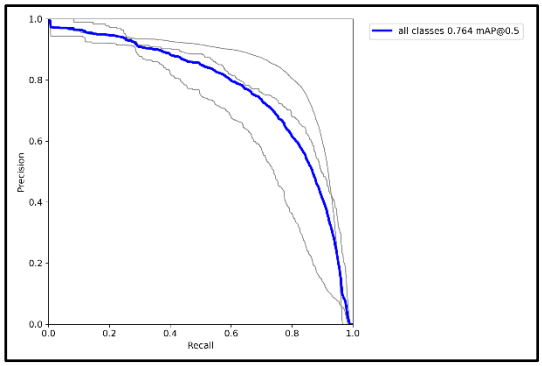


Figure 29 Ver2

從結果可以發現在 train 階段 mAP 的的值在 version2 有較好的表現，我認為是因為我分類資料夾較與選取照片的方式變成可以控制後，比較不會出現其中一個資料夾在 train_image_path 或 val_image_path 太少的導致 model 的預測不準的現象，在 test 時亦是如此。

與第一題相比 train 的 mAP 有了些微的下降了(0.977 降到 0.958)，撇除誤差的因素或是運氣的因素，我推測是因為 Q1 的訓練較專一，容易訓練出更高的 mAP，但這也是 Q1 的缺點，因為泛化性太低，所以在辨別沒看過的照片時就會失去訓練的效果，相反的，Q2 的結果雖然略低，但泛化性高，所以是更好的 model。

iii. Train F1 curve

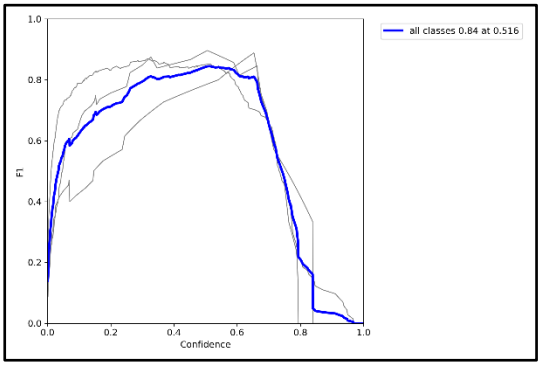


Figure 30 Ver1

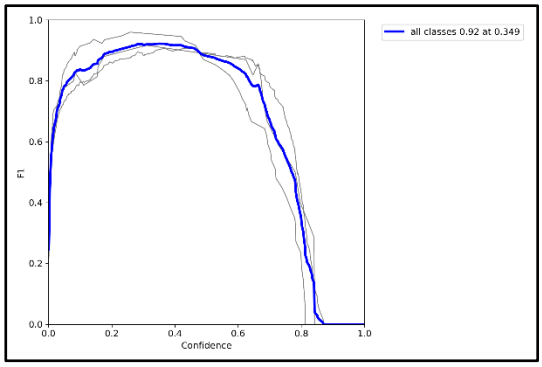


Figure 31 Ver2

iv. Test F1 curve

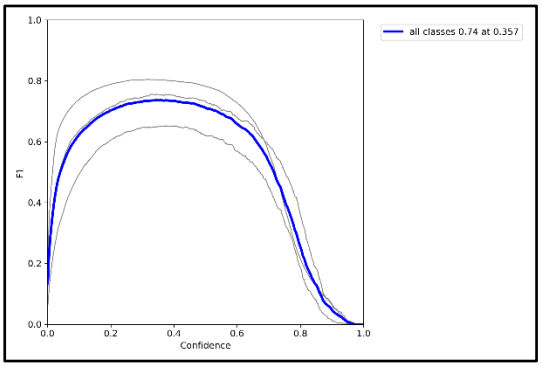


Figure 32 Ver1

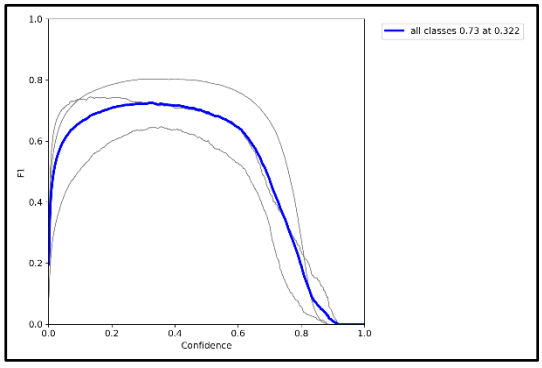


Figure 33 Ver2

在 test 的結果看起來 F1 curve 似乎沒甚麼差，這是可能是因為這次 ver1 的結果亂數的運氣很好才剛好與 ver2 接近，但在 training 階段就可以看出差異，F1 越接近 1 越好，可以看出 ver2 的值是較高的，與 Q1 相比會略低的原因，我認為是因為 Q1 的訓練較專一，代表 model 只要看懂兩個資料夾的 data 就可以 train 出好的 model，但 Q2 因為要看 6 個資料夾，泛化性變高了，model 必須認識更多特徵，那勢必更容易出錯，所以 train 的 mAP 就有些微下降，但這部分的影響不足以改變 model 因為泛化性提升帶來更好的 performance，除非選擇資料夾有極端的情形出現，否則 test 的結果還是優於 Q1。

Q3

In the final question, besides utilizing the dataset selected in Q2, you can incorporate another 1,200 unlabeled images. (45%)

I. Version1(add Q2 weight & freeze = 1 & change the hyp.scratch)

本小題的目的是將 unlabeled 的照片和 label 的照片 dataset 結合，希望可以降低人工 label 的時間成本來達到更好的訓練效果，我採取的方法是在 Q3 的資料夾隨機平均取出相同數量的照片共 200 張，取出的方式和 Q2 的 version1 相同，故在此不附上程式，而 labeled 照片我是將 Q2 訓練產生的 train 和 val 存有路徑的文字檔(txt)存起來，再用 with open 的方式引入，如下

```
with open('/content/drive/MyDrive/CV_hw4/Complete/Q2/train_Q2_ver2.txt', 'r') as file1:
    paths1 = file1.readlines()
with open('/content/drive/MyDrive/CV_hw4/Complete/Q2/val_Q2_ver2.txt', 'r') as file2:
    paths2 = file2.readlines()

Q2_select = paths1 + paths2
selected_image_paths.extend(Q2_select)
```

Figure 34 Select Code

Spilt code 也與 Q2 的 version2 相同，先將選得的路徑再依照資料夾分類，再分成 train 和 val。

```
def split_train_val_path(all_image_paths, train_val_ratio=0.9):
    # ver2
    import re
    folders = {'173': [], '398': [], '170': [], '410': [], '511': [], '495': []}
    # 提取文件路徑中的數字並進行分類
    for path in all_image_paths:
        match = re.search(r'/(\d+)-', path)
        if match:
            folder_number = match.group(1)
            if folder_number in folders:
                folders[folder_number].append(path)

    random.shuffle(folders['173'])
    camera_173_paths_train = folders['173'][: int(len(folders['173']) * train_val_ratio)]
    camera_173_paths_val = folders['173'][int(len(folders['173']) * train_val_ratio):]

    random.shuffle(folders['398'])
    camera_398_paths_train = folders['398'][: int(len(folders['398']) * train_val_ratio)]
    camera_398_paths_val = folders['398'][int(len(folders['398']) * train_val_ratio):]

    random.shuffle(folders['170'])
    camera_170_paths_train = folders['170'][: int(len(folders['170']) * train_val_ratio)]
    camera_170_paths_val = folders['170'][int(len(folders['170']) * train_val_ratio):]

    random.shuffle(folders['410'])
    camera_410_paths_train = folders['410'][: int(len(folders['410']) * train_val_ratio)]
    camera_410_paths_val = folders['410'][int(len(folders['410']) * train_val_ratio):]

    random.shuffle(folders['511'])
    camera_511_paths_train = folders['511'][: int(len(folders['511']) * train_val_ratio)]
    camera_511_paths_val = folders['511'][int(len(folders['511']) * train_val_ratio):]

    random.shuffle(folders['495'])
    camera_495_paths_train = folders['495'][: int(len(folders['495']) * train_val_ratio)]
    camera_495_paths_val = folders['495'][int(len(folders['495']) * train_val_ratio):]

    train_image_paths = camera_173_paths_train + camera_398_paths_train + camera_170_paths_train + camera_410_paths_train + camera_511_paths_train + camera_495_paths_train
    val_image_paths = camera_173_paths_val + camera_398_paths_val + camera_170_paths_val + camera_410_paths_val + camera_511_paths_val + camera_495_paths_val

    return train_image_paths, val_image_paths
```

Figure 35 Split code

```
# training command
!python train.py --project runs/train/Q3 --name refine --hyp '/content/drive/MyDrive/CV_hw4/CV2023_HW4/yolov7/data/hyp.scratch.custom.yaml' --freeze 1

--weights '/content/drive/MyDrive/CV_hw4/CV2023_HW4/yolov7/runs/train/Q2/refine2/weights/best.pt'
```

Figure 36 Train strategy

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
45/49     10.5G   0.03824   0.01642  0.008941    0.0636        32       640: 100% 30/30 [00:19<00:00,   1.54it/s]
          Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:00<00:00,   3.24it/s]
            all          42         182       0.624       0.775       0.647       0.417

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
46/49     10.5G    0.0353   0.01581   0.00792   0.05903        36       640: 100% 30/30 [00:20<00:00,   1.48it/s]
          Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:00<00:00,   3.37it/s]
            all          42         182       0.525       0.745       0.541       0.378

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     10.5G   0.03467   0.01567  0.007915   0.05825        23       640: 100% 30/30 [00:20<00:00,   1.44it/s]
          Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:00<00:00,   2.31it/s]
            all          42         182       0.542       0.737       0.542        0.37

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     10.5G   0.03551   0.01667  0.008114   0.06029        59       640: 100% 30/30 [00:20<00:00,   1.47it/s]
          Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:00<00:00,   3.35it/s]
            all          42         182       0.679       0.726       0.664       0.445

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     10.5G   0.03339   0.01664  0.007843   0.05787        34       640: 100% 30/30 [00:20<00:00,   1.45it/s]
          Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:01<00:00,   1.15it/s]
            all          42         182       0.678       0.675       0.677       0.441
```

Figure 37 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.611	0.625	0.648	0.429
	car	1200	1504	0.908	0.783	0.884	0.543
	bus	1200	210	0.749	0.795	0.862	0.62
	truck	1200	47	0.177	0.298	0.197	0.123
495							
	all	1200	2227	0.686	0.793	0.754	0.483
	car	1200	1950	0.695	0.881	0.788	0.488
	bus	1200	55	0.877	0.778	0.874	0.629
	truck	1200	222	0.487	0.721	0.599	0.332
410							
	all	1200	2331	0.711	0.755	0.755	0.506
	car	1200	2005	0.882	0.689	0.801	0.509
	bus	1200	15	0.418	0.8	0.627	0.506
	truck	1200	311	0.834	0.777	0.836	0.503
511							
	all	1200	2294	0.726	0.676	0.697	0.49
	car	1200	2077	0.901	0.744	0.854	0.516
	bus	1200	86	0.849	0.849	0.873	0.716
	truck	1200	131	0.429	0.435	0.365	0.239
398							
	all	1200	2353	0.715	0.734	0.751	0.513
	car	1200	2056	0.731	0.796	0.77	0.489
	bus	1200	104	0.906	0.557	0.791	0.591
	truck	1200	193	0.507	0.85	0.694	0.461
173							
	all	1200	1991	0.819	0.675	0.81	0.558
	car	1200	1680	0.88	0.817	0.909	0.609
	bus	1200	171	0.867	0.608	0.814	0.578
	truck	1200	140	0.711	0.6	0.707	0.486
ALL							
	all	1200	12957	0.746	0.698	0.754	0.501
	car	1200	11272	0.835	0.758	0.817	0.514
	bus	1200	641	0.794	0.667	0.803	0.591
	truck	1200	1044	0.609	0.67	0.642	0.398

Figure 38 Test results

加入 freeze=1 的用意我認為和 dropout 的功能是類似的，等於一代表可以使 model 的第一層不訓練，可以藉此降低 overfitting 的產生。

加入 Q2 的 weight 是為了讓 model 有一個 pre train 的權重，藉此提高訓練效果。

從結果上來說加入 unlabeled 照片沒有有效提升 mAP，最多就是和 Q2 的 mAP 相同，我推測是因為如果只改這些 hyperparameter 是不足以好好利用 Q3 的 unlabeled 照片，因為對於訓練來說，這些 unlabeled 照片再訓練裡可能就像白紙，會擾亂訓練的精準度。

i. PR curve

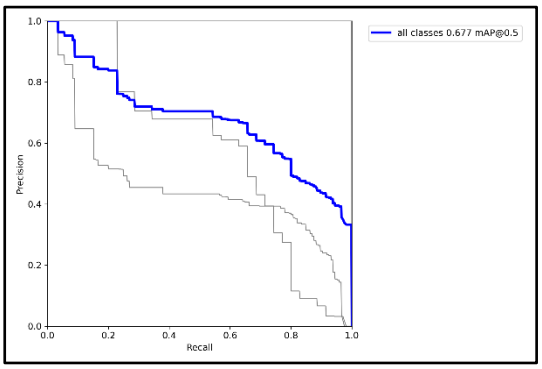


Figure 39 Train

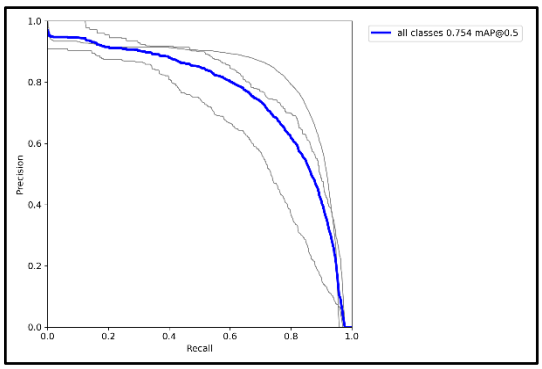


Figure 40 Test

ii. F1 curve

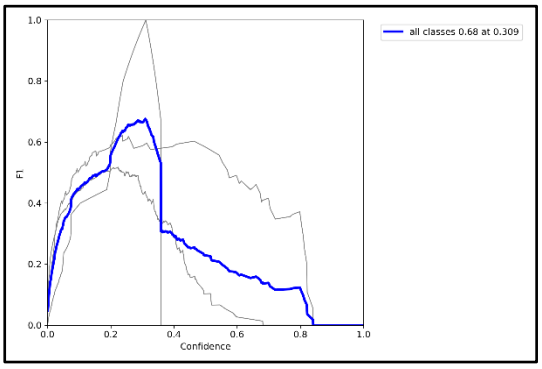


Figure 41 Train

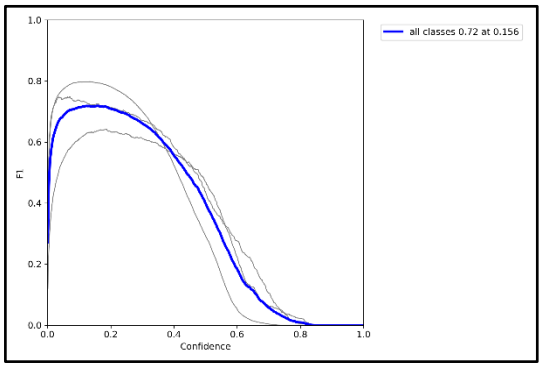


Figure 42 Test

從 PR curve 可以發現在 train 的 mAP 下降了許多，但在 test 的結果還是有一定的效果，所以我推測 train 會下降是因為 unlabel 的照片在拉低 precision 的值，因為訓練時 model 看過更多照片，所以理論上他會更泛化，所以 recall 值應該會上升，model 容易把該識別的物品辨識出來，但正確率也會因此下降。

F1 curve 也和 Q1,Q2 的結果相反，test 也比 train 高，原因我認為和 PR curve 是相同的；一般來說 F1 curve 應該有一個較穩定的高點 F1 值，但這個 train 的結果沒有這個現象，代表這個訓練策略可能是不適合的。

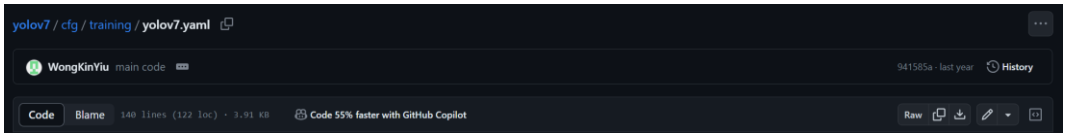
II. **Version2**(add Q2 weight & freeze = 1 & cfg)

```
[ ]  # training command
     !python train.py --project runs/train/Q3 --name refine --hyp '/content/drive/MyDrive/CV_hw4/CV2023_HW4/yolov7/data/hyp.scratch.p5.yaml' --freeze 1

--weights '/content/drive/MyDrive/CV_hw4/CV2023_HW4/yolov7/runs/train/Q2/refine2/weights/best.pt' --cfg '/content/drive/MyDrive/CV_hw4/CV2023_HW4/yolov7/yolov7.yaml'
```

Figure 43 Train strategy

因為在 train 產生的 yaml 檔發現 cfg 欄是空白的，故嘗試加入提高 mAP，cfg 是 YOLOv7 的官方項目提供了預先訓練好 pretrained model，可以在 GitHub 找到 source code。




```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
47/49     11.1G   0.02877   0.01239  0.003399   0.04456        25       640: 100% 30/30 [00:22<00:00,  1.33it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00,  2.15it/s]
             all         42        182      0.609      0.689      0.651      0.473

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
48/49     11.1G   0.02759   0.01201  0.003096   0.04277        86       640: 100% 30/30 [00:21<00:00,  1.40it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:00<00:00,  3.22it/s]
             all         42        182      0.649      0.902      0.777      0.536

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
49/49     11.1G   0.02765   0.01198  0.003811   0.04344        44       640: 100% 30/30 [00:21<00:00,  1.37it/s]
           Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.94it/s]
             all         42        182      0.674      0.893      0.828      0.563
```

Figure 44 Train result

Camera	Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
170							
	all	1200	1761	0.624	0.463	0.523	0.329
	car	1200	1504	0.805	0.566	0.728	0.415
	bus	1200	210	0.801	0.652	0.716	0.491
	truck	1200	47	0.265	0.17	0.126	0.082
495							
	all	1200	2227	0.698	0.591	0.657	0.39
	car	1200	1950	0.703	0.587	0.683	0.407
	bus	1200	55	0.804	0.595	0.676	0.454
	truck	1200	222	0.588	0.59	0.611	0.309
410							
	all	1200	2331	0.608	0.397	0.477	0.302
	car	1200	2005	0.75	0.386	0.573	0.361
	bus	1200	15	0.313	0.333	0.256	0.193
	truck	1200	311	0.762	0.473	0.603	0.352
511							
	all	1200	2294	0.541	0.502	0.501	0.324
	car	1200	2077	0.607	0.458	0.531	0.288
	bus	1200	86	0.652	0.674	0.628	0.466
	truck	1200	131	0.364	0.374	0.343	0.219
398							
	all	1200	2353	0.741	0.567	0.675	0.448
	car	1200	2056	0.754	0.417	0.61	0.345
	bus	1200	104	0.96	0.46	0.628	0.452
	truck	1200	193	0.509	0.824	0.786	0.548
173							
	all	1200	1991	0.626	0.545	0.563	0.342
	car	1200	1680	0.698	0.618	0.683	0.404
	bus	1200	171	0.747	0.553	0.587	0.366
	truck	1200	140	0.435	0.464	0.417	0.255
ALL							
	all	1200	12957	0.646	0.526	0.579	0.362
	car	1200	11272	0.695	0.491	0.616	0.357
	bus	1200	641	0.774	0.55	0.617	0.422
	truck	1200	1044	0.469	0.538	0.505	0.306

Figure 45 Test results

從 train 結果可以發現使用 cfg 後的訓練 mAP 上升了許多，但 test 的結果卻不是很理想。

i. PR curve

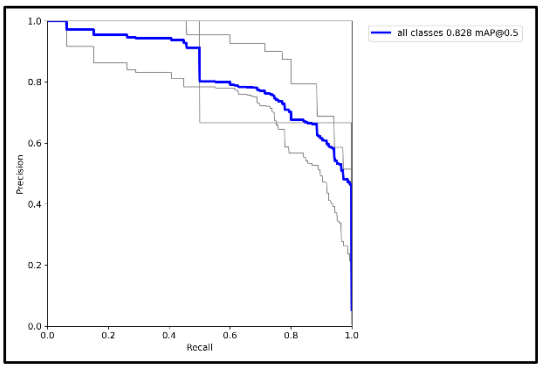


Figure 46 Train

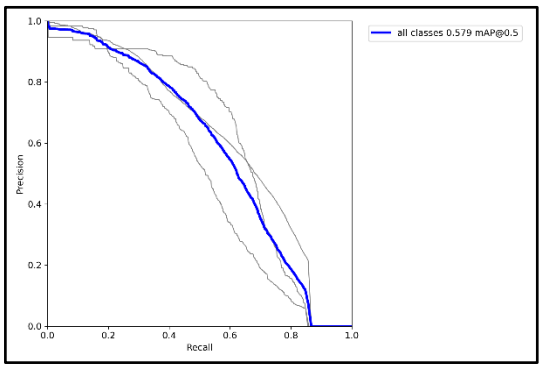


Figure 47 Test

ii. F1 curve

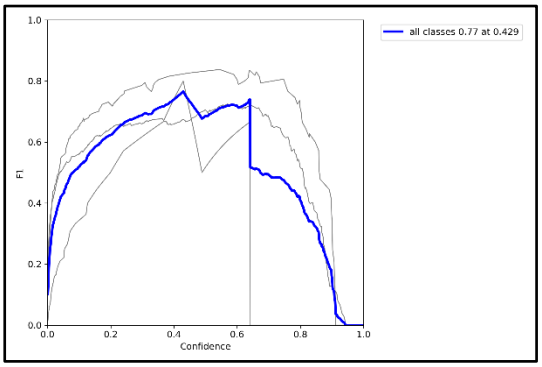


Figure 48 Train

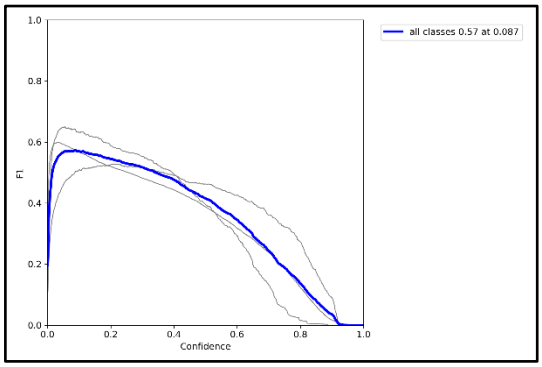


Figure 49 Test

從 PR curve 也可以發現使用 cfg 後的訓練 mAP 上升了許多，但 test 下降許多，我推測是因為使用 cfg 後訓練有 overfitting 的現象，所以才會訓練好，測試差，而造成 overfitting 原因我有測試了幾種可能性，第一種是 split 的 ratio，從 0.9 降到 0.8，透過增加 validation 的資料，讓 model 可以有較好的驗證，但這個方法似乎不能改善；第二種方法，我調整了 hyperparameter 中的 learn rate，從 0.01 變 0.001，但結果也不如預期；第三種我調整了 model.gr，這是可以調整預測時 IoU 的大小，當設 1 時代表預測框框與實際框框要重疊才會判斷是正確，越低兩個框框的重疊面積越小，但不代表變差，雖然 precision 的值可能下降，但 recall 值理論上會上升，題目一開始預設的 model.gr = 1，因為加入 unlabeled 照片，精確度本來就會下降，要提升的話我認為可以用較不嚴謹的方法先預測，故我將其改成 0.1 實測，但此法也未解決 overfitting 的問題；第四種我調整降低了 label smoothing，這也是讓預測不要太嚴謹的方法，overfitting 依然存在。

III. Discussion

在 Q3 我討論了好幾種調整的方法，但都無法有效的提升 mAP，我認為是因為我沒好好運用 unlabeled 照片的原因，理論上 label 的照片一定比 unlabeled 的照片有更好的訓練結果，但只透過調整 hyperparameter 或是訓練權重，在 label 照片的訓練上會有改善，但 unlabeled 可能就無法了。

在題目上還有其他方法，例如 pseudo label，這是一種半監督是學習，這是將 unlabeled 照片先用 label 照片訓練完的結果對其預測，產生 pseudo label，再將 pseudo label 和 label 合在一起進行訓練，這個方法就有確保使用到 unlabeled 照片，雖然我沒有實測過，但我認為這應該就可以有效提升 mAP，在提升的過程也可以減少人工 label 的時間成本。