圖形識別期末報告

電機碩一_7112064005_蔡雨竹

[主題簡介]

本研究旨在比較和評估多種主流物體偵測模型在 Kitti 數據集上的性能。我選擇了三種 One-stage 模型（YOLOv3、YOLOv4 和 RetinaNet）以及兩種 Two-stage 模型（Faster R-CNN 和 Mask R-CNN），對它們的平均精確度（mAP）和測試時間（ms）進行分析。

[方法敘述]

1. Dataset

   1.1 YOLOv3、RetinaNet、Faster R-CNN、Mask R-CNN

   為了更準確地評估模型性能，我將 Kitti 數據集劃分為訓練集（圖像 001496-006283，共 4788 張）、驗證集（圖像 006284-007480，共 1197 張）和測試集（圖像 000000-001495，共 1496 張）。比例為訓練集: 64%，驗證集: 16%，測試集: 20%。

   1.2 YOLOv4

   這邊的 dataset 我分為 train(圖像 000000-004499，共 4500 張):val（圖像 004500-005995，共 1496 張）:test(圖像 005996-007480，共 1485 張) 比例為訓練集: 60%，驗證集: 20%，測試集: 20%。 因為想試試看會跑出甚麼樣結果，所以稍微變動一下資料集。

2. YOLOv3

   使用課堂 Lab2 Object Detection 上所學的 PyTorch-YOLOv3-kitti-master 進行訓練

3. YOLOv4

   使用 https://github.com/AlexeyAB/darknet

   指令如下:

   # Training

   darknet detector train C:/Proposal/darknet/build/darknet/x64/data/kitty.data C:/Proposal/darknet/build/darknet/x64/cfg/yolov4_kitty.cfg C:/Proposal/Stanford_Dogs_Dataset/Trainingfiles/yolov4.conv.137 -mjpeg_port 8090 -map -gpus 0 -dont_show

   # map

   darknet detector map C:/Proposal/darknet/build/darknet/x64/data/kitty.data C:/Proposal/darknet/build/darknet/x64/cfg/yolov4_kitty.cfg C:/Proposal/darknet/build/darknet/x64/backup/yolov4_kitty_best.weights

Test

python patch_test.py

C:/Proposal/darknet/build/darknet/x64/cfg/yolov4_kitty.cfg

C:/Proposal/darknet/build/darknet/x64/backup/yolov4_kitty_best.weights

C:/kitty/yolov4/test.txt

由於 github 包裡面的測試只能輸入單張照片，我額外寫了一個程式檔 patch_test.py（如附檔所示），可以進行多張照片測試。

4. RetinaNet、Faster R-CNN、Mask R-CNN

使用 detectron2（https://github.com/facebookresearch/detectron2）來進行訓練。需注意兩個部分：

4.1 資料準備

由於訓練檔格式不同，需要轉換成.json 檔。我編寫 modify.py（如附檔所示），只需在程式碼中放入圖片位置與標籤位置即可生成正確的 json 檔案。

4.2 訓練檔與測試檔

我另外寫了一個 train.py（如附檔所示），使程式碼可以進行訓練和驗證。訓練部分只需更改以下兩個地方即可運行不同模型。訓練指令為：python train.py --num-gpus 1

a. 配置文件 args.config_file =

```
def setup(args):
    # 註冊資料集
    plain_register_dataset()

    cfg = get_cfg()
    args.config_file = "E:/detectron2/configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"
    # RetinaNet:args.config_file = "E:/detectron2/configs/COCO-Detection/retinanet_R_50_FPN_3x.yaml"
    # Faster R-CNN:args.config_file = "E:/detectron2/configs/COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"
    # Mask R-CNN: args.config_file = "E:/detectron2/configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"
```

圖 1 配置文件

b. 模型權重 cfg.MODEL.WEIGHTS =

```
cfg.MODEL.WEIGHTS = "E:/detectron2/hmw/model_final_f10217.pkl"
# RetinaNet cfg.MODEL.WEIGHTS = "E:/detectron2/hmw/model_final_5bd44e.pkl"
# Faster R-CNN cfg.MODEL.WEIGHTS ="E:/detectron2/hmw/model_final_280758.pkl"
# Mask R-CNN cfg.MODEL.WEIGHTS = "E:/detectron2/hmw/model_final_f10217.pkl"
```

圖 2 模型權重

此外，如果進行測試，只需將 VAL_PATH 和 VAL_JSON 更改為存放的測試位置即可。測試指令為：: python train.py --eval-only

```
# 設定資料集路徑
DATASET_ROOT = 'E:/detectron2/datasets/kitti'
TRAIN_PATH = os.path.join(DATASET_ROOT, 'images/train')
VAL_PATH = os.path.join(DATASET_ROOT, 'images/val')
# VAL_PATH = os.path.join(DATASET_ROOT, 'images/test')

TRAIN_JSON = os.path.join(DATASET_ROOT, 'json/train/train_corrected.json')
VAL_JSON = os.path.join(DATASET_ROOT, 'json/val/val_corrected.json')
# VAL_JSON = os.path.join(DATASET_ROOT, 'json/test/test_corrected.json')
```

圖 3 測試文件路徑

[結果比較及討論]

Table 1 各模型平均精確度比較

Model	AP(Car)	AP(Van)	AP(Truck)	AP(Pedestrian)	AP(Person_sitting)	AP(Cyclist)	AP(Tram)	AP(Misc)	mAP (IOU=.50)
YOLO V3	92.89	93.7	97.01	82.03	74.99	90.15	94.26	82.78	88.48
YOLO V4	89.08	83.92	81.76	45.03	20.66	53.1	59.55	65.76	62.36
RetinaNet	61.062	52.34	66.854	28.735	28.157	34.596	58.164	37.776	72.5
Faster R-CNN	66.491	59.916	73.696	33.094	24.446	42.318	51.284	47.78	79.68
Mask R-CNN	67.95	60.274	74.054	34.443	29.081	42.14	50.031	49.506	79.58

Table 2 各模型在 AP50、AP75 等指標上的表現

Model	AP@.50	AP@.75	AP@[.5:.05:.95]	APS	APM	APL	Test time	Environment
YOLO V3	88.48	80.15	85.89	74.84	83.56	90.11	452.22s	Intel i7, 32GB RAM
YOLO V4	62.36	42.53	39.37	0.305987	0.397846	0.504789	19s	NVIDIA GeForce RTX 3080, 8GB RAM
RetinaNet	72.459	49.094	45.9604	27.408	43.808	63.241	36.29s	NVIDIA GeForce RTX 2070, 32GB RAM
Faster R-CNN	79.681	53.687	49.878	49.743	47.636	59.2	38.06s	NVIDIA GeForce RTX 2070, 32GB RAM
Mask R-CNN	79.587	55.653	50.828	49.683	48.982	59.625	67s	NVIDIA GeForce RTX 2070, 32GB RAM

1. YOLOv3

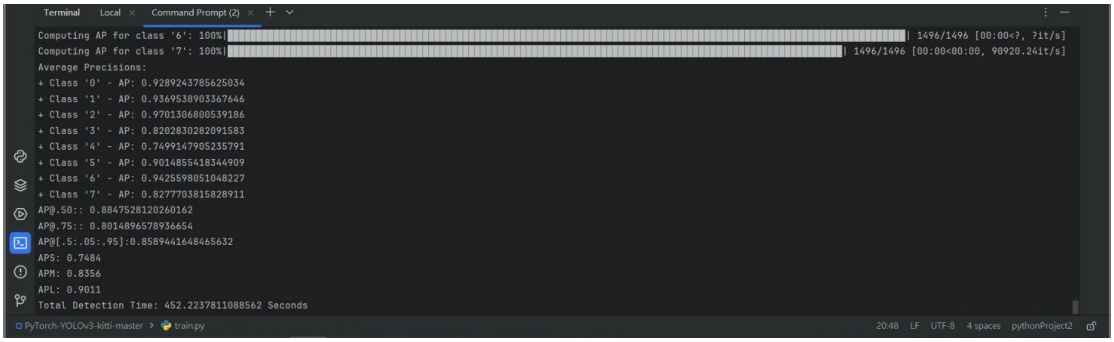

圖 4 測試結果圖

圖 5 000006.png

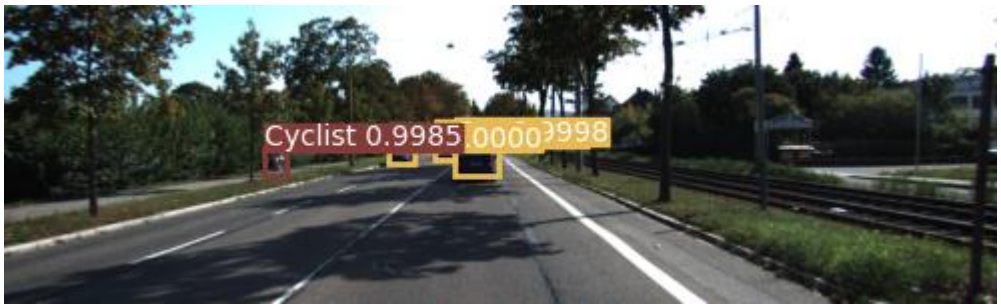

圖 6 000007.png

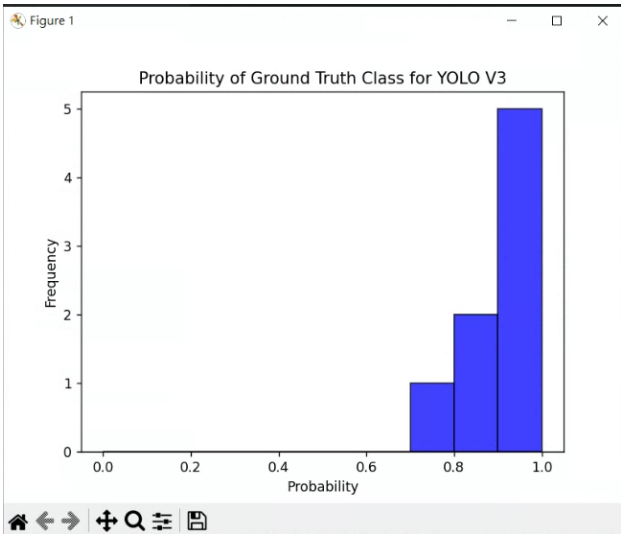

圖 7 YOLO V3 Ground Truth Class Probability Distribution

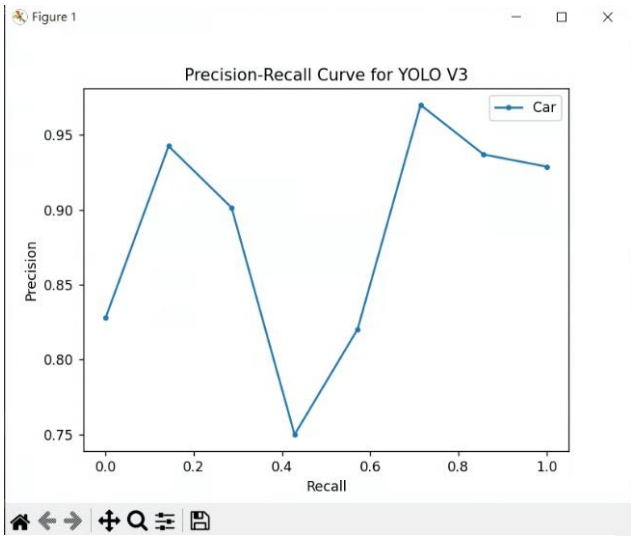

圖 8 YOLO V3 Precision-Recall Curve

2. YOLOv4

```
detections_count = 10435, unique_truth_count = 8343
class_id = 0, name = Car, ap = 89.08%          (TP = 4992, FP = 304)
class_id = 1, name = Van, ap = 83.92%          (TP = 449, FP = 39)
class_id = 2, name = Truck, ap = 81.76%        (TP = 167, FP = 28)
class_id = 3, name = Pedestrian, ap = 45.03%   (TP = 313, FP = 71)
class_id = 4, name = Person_sitting, ap = 20.66%        (TP = 2, FP = 0)
class_id = 5, name = Cyclist, ap = 53.10%      (TP = 148, FP = 39)
class_id = 6, name = Tram, ap = 59.55%         (TP = 44, FP = 5)
class_id = 7, name = Mis, ap = 65.76%          (TP = 94, FP = 14)

 for conf_thresh = 0.25, precision = 0.93, recall = 0.74, F1-score = 0.83
 for conf_thresh = 0.25, TP = 6209, FP = 500, FN = 2134, average IoU = 74.34 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.623567, or 62.36 %
 IoU threshold = 75 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.75) = 0.425298, or 42.53 %
 AP = 0.393655, or  39.37%
 APS = 0.305987
 APM = 0.397846
 APL = 0.504789

Total Detection Time: 19 Seconds
>>> _
```

圖 9 YOLOv4 測試結果圖

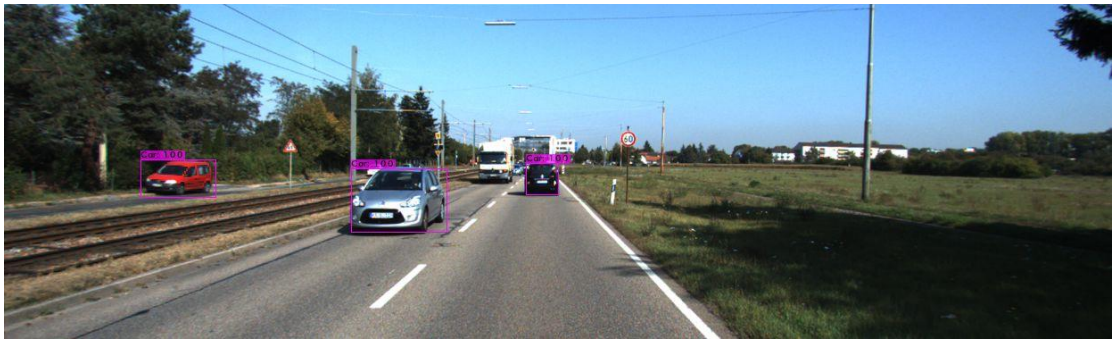

圖 10 005998.png

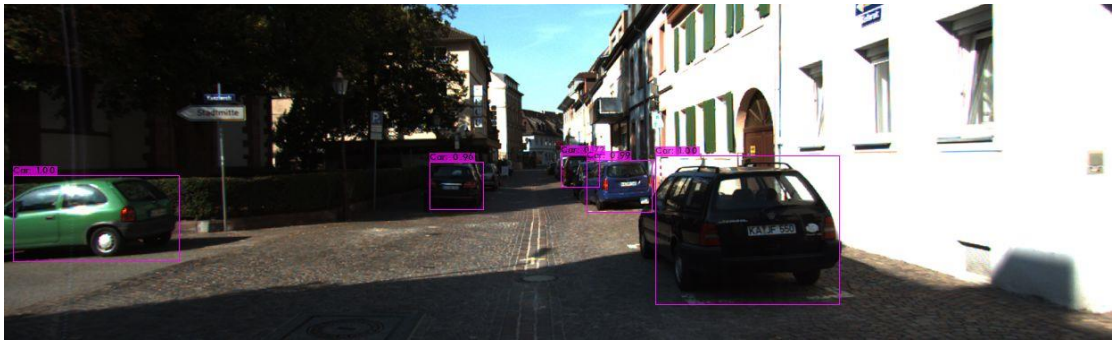

圖 11 005999.png

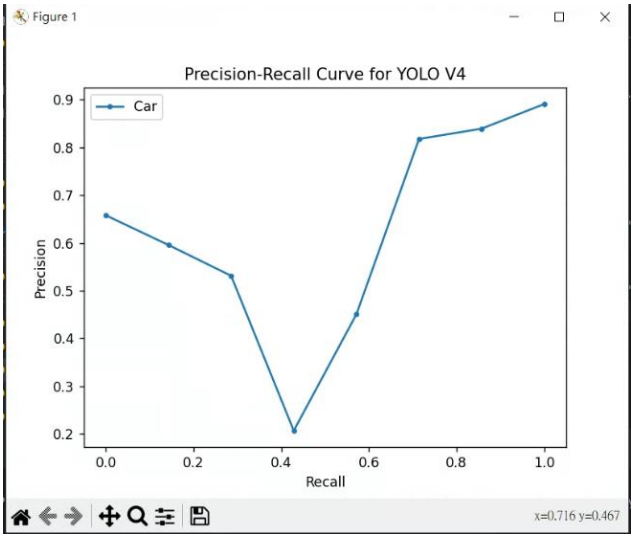

圖 12 YOLO V4 Precision-Recall Curve

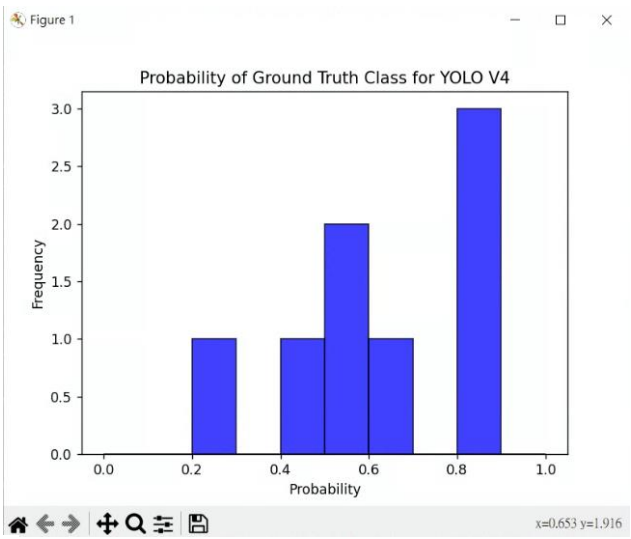

圖 13 YOLO V4 Ground Truth Class Probability Distribution

3. RetinaNet

```
[06/15 02:50:08 d2.evaluation.evaluator]: Total inference time: 0:00:36.298935 (0.030452 s / iter per device, on 1 devices)
[06/15 02:50:08 d2.evaluation.evaluator]: Total inference pure compute time: 0:00:35 (0.029397 s / iter per device, on 1 devices)
[06/15 02:50:10 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.15 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.460
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.725
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.491
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.274
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.438
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.632
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.352
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.566
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.587
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.444
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.566
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.722
[06/15 02:50:10 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 45.960 | 72.459 | 49.094 | 27.408 | 43.808 | 63.241 |
[06/15 02:50:10 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category       | AP     | category   | AP     |
|:-----------|:-------|:---------------|:-------|:-----------|:-------|
| Car        | 61.062 | Van            | 52.340 | Truck      | 66.854 |
| Pedestrian | 28.735 | Person_sitting | 28.157 | Cyclist    | 34.596 |
| Tram       | 58.164 | Misc           | 37.776 |            |        |
[06/15 02:50:10 d2.engine.defaults]: Evaluation results for coco_my_val in csv format:
[06/15 02:50:10 d2.evaluation.testing]: copypaste: Task: bbox
[06/15 02:50:10 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[06/15 02:50:10 d2.evaluation.testing]: copypaste: 45.9604,72.4590,49.0943,27.4081,43.8079,63.2409
```

圖 14 RetinaNet 測試結果圖

圖 15 000006.png

圖 16 000007.png

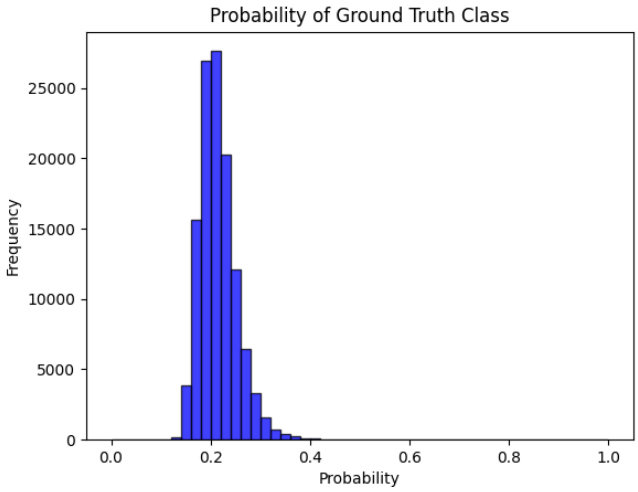

圖 17 RetinaNet Ground Truth Class Probability Distribution

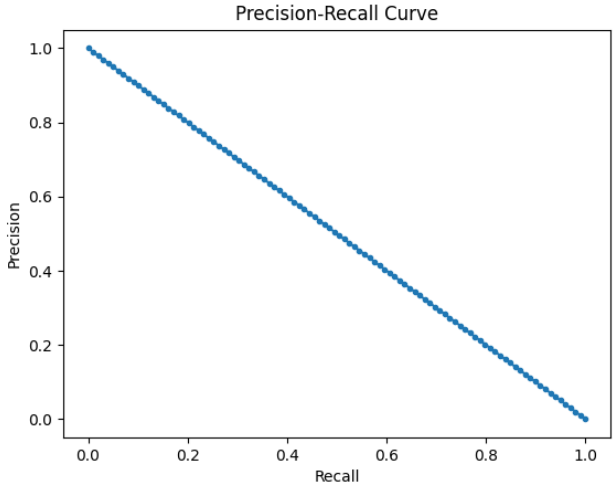

圖 18 RetinaNet Precision-Recall Curve

4. Faster R-CNN

```
[06/15 15:06:39 d2.evaluation.evaluator]: Total inference time: 0:00:38.064894 (0.031934 s / iter per device, on 1 devices)
[06/15 15:06:39 d2.evaluation.evaluator]: Total inference pure compute time: 0:00:36 (0.030922 s / iter per device, on 1 devices)
[06/15 15:06:39 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.05 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.499
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.797
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.537
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.497
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.476
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.592
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.380
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.585
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.594
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.591
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.565
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.679
[06/15 15:06:39 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 49.878 | 79.681 | 53.687 | 49.743 | 47.636 | 59.200 |
[06/15 15:06:39 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category       | AP     | category   | AP     |
|:-----------|:-------|:---------------|:-------|:-----------|:-------|
| Car        | 66.491 | Van            | 59.916 | Truck      | 73.696 |
| Pedestrian | 33.094 | Person_sitting | 24.446 | Cyclist    | 42.318 |
| Tram       | 51.284 | Misc           | 47.780 |            |        |
[06/15 15:06:39 d2.engine.defaults]: Evaluation results for coco_my_val in csv format:
[06/15 15:06:39 d2.evaluation.testing]: copypaste: Task: bbox
[06/15 15:06:39 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[06/15 15:06:39 d2.evaluation.testing]: copypaste: 49.8779,79.6808,53.6873,49.7430,47.6356,59.2001
```

圖 19 Faster R-CNN 測試結果圖

圖 20 000006.png

圖 21 000007.png

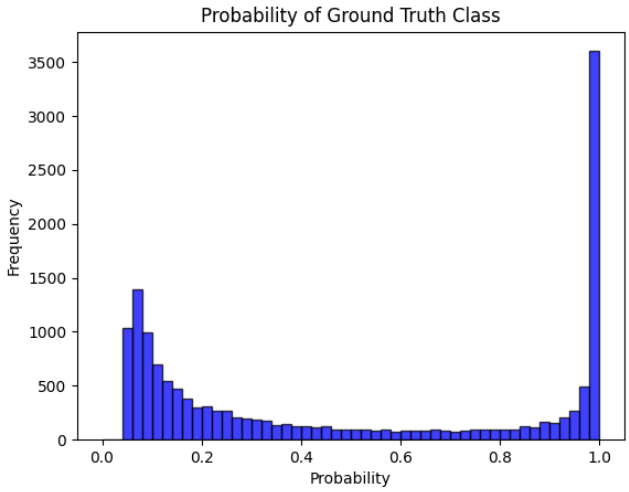

圖 22 Faster R-CNN Ground Truth Class Probability Distribution

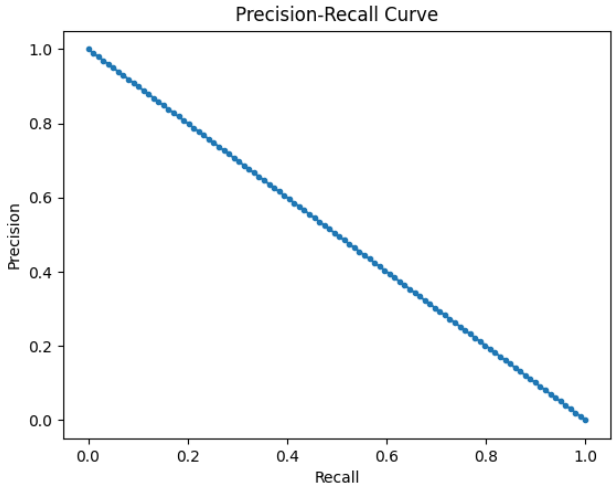

圖 23 Faster R-CNN Precision-Recall Curve

Mask R-CNN

```
[06/15 05:59:21 d2.evaluation.evaluator]: Total inference time: 0:01:07.
[06/15 05:59:21 d2.evaluation.evaluator]: Total inference pure compute t
```

```
[06/15 05:59:22 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.04 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.508
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.796
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.557
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.497
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.490
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.596
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.387
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.591
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.599
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.597
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.577
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.673
[06/15 05:59:22 d2.evaluation.coco_evaluation]: Evaluation results for segm:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 50.828 | 79.587 | 55.653 | 49.683 | 48.982 | 59.625 |
[06/15 05:59:22 d2.evaluation.coco_evaluation]: Per-category segm AP:
| category   | AP     | category       | AP     | category   | AP     |
|:-----------|:-------|:---------------|:-------|:-----------|:-------|
| Car        | 67.095 | Van            | 60.274 | Truck      | 74.054 |
| Pedestrian | 34.443 | Person_sitting | 29.081 | Cyclist    | 42.140 |
| Tram       | 50.031 | Misc           | 49.506 |            |        |
[06/15 05:59:22 d2.engine.defaults]: Evaluation results for coco_my_val in csv format:
[06/15 05:59:22 d2.evaluation.testing]: copypaste: Task: bbox
[06/15 05:59:22 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[06/15 05:59:22 d2.evaluation.testing]: copypaste: 51.7095,80.2719,57.4482,49.2941,49.4926,60.9149
[06/15 05:59:22 d2.evaluation.testing]: copypaste: Task: segm
[06/15 05:59:22 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl
[06/15 05:59:22 d2.evaluation.testing]: copypaste: 50.8281,79.5874,55.6534,49.6832,48.9818,59.6246
```

圖 24 Mask R-CNN 測試結果圖

預測圖:

圖 25 000006.png

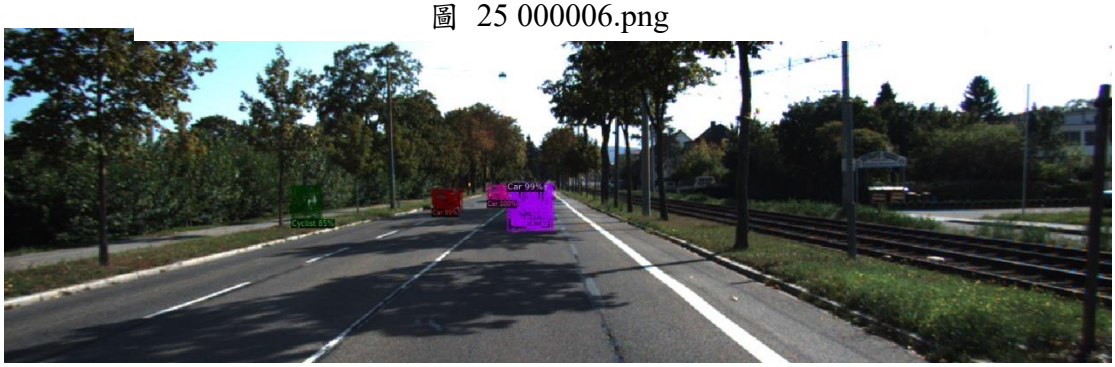

圖 26 000007.png

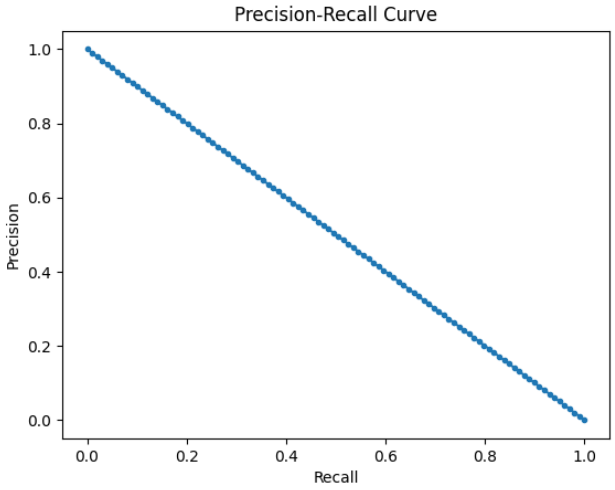

圖 27 Mask R-CNN Precision-Recall Curve

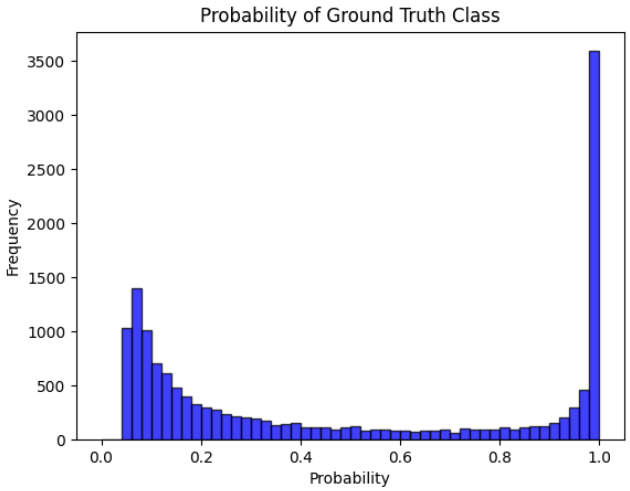

圖 28 Mask R-CNN Ground Truth Class Probability Distribution

[各方面比較]

6. 平均精確度 (Average Precision, AP) 比較

6.1 YOLOv3 表現最佳，特別是在 Car (92.89)、Van (93.7)、Truck (97.01) 和 Cyclist (90.15) 類別上，其 mAP (IOU=0.5) 達到 88.48。

6.2 YOLOv4 的整體表現較弱，特別是在 Person_sitting (20.66) 和 Pedestrian (45.03) 類別上，顯示出該模型在行人檢測方面的劣勢。

6.3 RetinaNet 在 Car (61.062) 和 Pedestrian (28.735) 類別上的表現較差，整體 mAP (IOU=0.5) 為 48.6。

6.4 Faster R-CNN 和 Mask R-CNN 的性能較為相似，在 Truck、Cyclist 和 Tram 類別上表現優異，mAP 分別為 79.681 和 79.587。

7. AP50 和 AP75 比較

7.1 YOLOv3 的 AP50 和 AP75 分別為 88.48 和 80.15，整體表現優異。

7.2 YOLOv4 在這兩個指標上的表現較差，分別為 62.36 和 42.53。

7.3 RetinaNet 的 AP50 為 72.459，AP75 為 49.094，顯示其在較高 IOU 閾值下性能有所下降。

7.4Faster R-CNN 和 Mask R-CNN 的 AP50 和 AP75 指標相對較高，顯示出這兩個模型在不同 IOU 閾值下的穩定性。

8. 不同大小目標的檢測能力 (APS, APM, APL)

8.1 YOLOv3 在檢測小目標（APS）方面的表現最優，達到 74.84，而在中目標（APM）和大目標（APL）上的表現也較為出色，分別為 83.56 和 90.11。

8.2 YOLOv4 在小目標檢測上表現最差，APS 為 0.305987，顯示出該模型在小目標檢測上的劣勢。

8.3 RetinaNet 在小目標檢測上的表現相對較好，APS 為 27.408。

8.4 Faster R-CNN 和 Mask R-CNN 在不同大小目標上的檢測能力較為均衡。

9. 測試時間 (Test time)

9.1 YOLOv3 的測試時間為 452.22 毫秒，相對較慢，但考慮到其優異的檢測性能，這一點可以接受。

9.2 YOLOv4 的測試時間最短，僅為 19 秒，顯示出該模型在速度上的優勢。

9.3 RetinaNet、Faster R-CNN 和 Mask R-CNN 的測試時間分別為 36.29 毫秒、38.06 毫秒和 67 毫秒，表現適中。

[心得與結論]

我覺得 YOLOv3 跟 YOLOv4 因為在先前課程有先碰過所以比較熟悉一點，唯一遇到的挑戰是 APS、APM、APL，原本的訓練檔格式需要轉換成 JSON 格式才能夠跑出，所以我又重新訓練一次。RetinaNet、Faster R-CNN 和 Mask R-CNN 我使用 detectron2 來做訓練，我遇到最大的挑戰是電腦編譯不起來，所以我用 cpu-pytorch 來做訓練，但是發現訓練要 1 天多，我接著改用筆電安裝，但我的筆電是 RTX 1650 訓練也要 12 個多小時。後來發現我的電腦所有路徑接設定正確，所以我直接進到 setup.py 加入以下#'-allow-unsupported-compiler', 順利安裝，而且 torch 也相容，電腦使用 RTX 2070 只需要 2 個多小時即可訓練完成。雖然在上述結果比較中，YOLOv3 在大部分測試中表現比較好，但是他的訓練時間整整耗費 1 天多的時間，YOLOv4 甚至跑了快 3 天，YOLOv4 雖然測試速度上有明顯優勢，但在檢測性能方面仍需改進。RetinaNet、Faster R-CNN 和 Mask R-CNN 則在不同大小目標檢測上表現均衡，適合多種應用場景。

[參考文獻]

[1] https://github.com/AlexeyAB/darknet.git

[2] https://github.com/facebookresearch/detectron2

[3] https://blog.csdn.net/qq_29750461/article/details/106761382

[4] https://blog.csdn.net/weixin_41655296/article/details/118720338