

# **Autonomous Driving Final Project**

## **-- 2D object detection**

Group 13

電機系大四 E24074067 陳彥銘

電機系大四 E24076467 黃瑋翔

電機系大四 E24073053 陳祈佑

電機所碩一 N26104337 李惠軒

電機所碩一 N26114277 卓冠廷

電機所碩一 N26100016 張智彥

<b>1. Introduction</b>	<b>3</b>
<b>2. Related Work</b>	<b>3</b>
<b>2.1 Dataset Introduction</b>	<b>3</b>
<b>2.2 Data augmentation</b>	<b>4</b>
<b>2.3 Object detection</b>	<b>6</b>
<b>2.3.1 One-Stage</b>	<b>6</b>
<b>2.3.2 Two-Stage</b>	<b>7</b>
<b>3. Methods</b>	<b>8</b>
<b>3.1 YOLO</b>	<b>8</b>
<b>3.2 YOLO v1:</b>	<b>8</b>
<b>3.3 YOLO v2:</b>	<b>8</b>
<b>3.4 YOLO v3:</b>	<b>8</b>
<b>3.5 YOLO v4:</b>	<b>14</b>
<b>3.6 YOLO v5:</b>	<b>16</b>
<b>4. Result(YOLO 實作)</b>	<b>19</b>
<b>4.1 YOLO v3 vs. YOLO v4</b>	<b>19</b>
<b>4.2 YOLO v5</b>	<b>27</b>
<b>5. Conclusion</b>	<b>29</b>
<b>6. Reference</b>	<b>32</b>

# 1. Introduction

近年來許多車廠相繼開發的自動駕駛系統，無非是希望能開發出一套辨識效果好、靈敏性高且安全度極高的自動駕駛系統，而最基本的即是準確的辨識車體周遭的行人及車輛等，以利系統做出相對應的反應。

Waymo 發布開放數據，以幫助研究智能感應和自動駕駛技術的發展。2022 年 Waymo 將感應數據擴展為包括關鍵標籤、2D 到 3D 關鍵標籤和 3D 分割標籤。條件的多樣性：市區、郊區、白天、黑夜、行人、自行車騎士、建築與各式各樣的天氣。

Waymo Open Dataset 大規模且多樣性高的數據集提供人們開發自動駕駛系統，在多種場景取得的數據集，內部包含日期、氣候、以及多個高分辨率攝影機生成的影像及雷達從傳感器讀取的數值。數據集共有 1150 種場景，1000 個場景用於 training 及 validation、150 個場景用於 testing，而每個場景包含了 200 frames，1 frame 有 5 個視角。資料集內部共標示了 5 種類別：

'TYPE\_UNKNOWN','TYPE\_VECHICLE','TYPE\_PEDESTRIAN','TYPE\_SIGN',  
'TYPE\_CYCLIST'。

本報告的研究目標為 2D Detection，使用數據集 Waymo Open Dataset，比較不同版本 YOLO 的效果差異，並透過資料增強以強化資料量較小的類別以提高訓練成果。

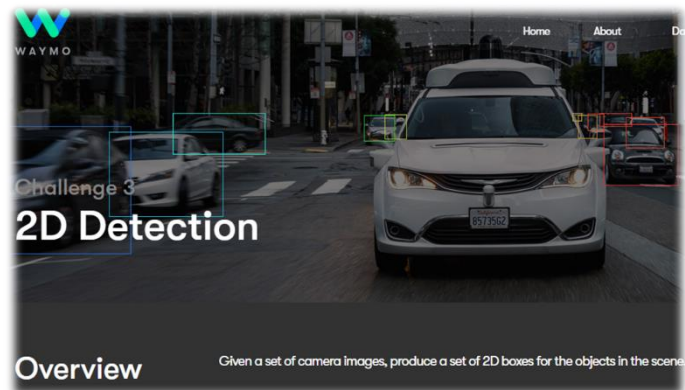


Fig. 1 Waymo Open Dataset 2D Detection [1]

## 2. Related Work

### 2.1 Dataset Introduction

Waymo 於 2019 年 7 月公開了自動駕駛數據集 Waymo Open Dataset[1]，是該領域品質最高、規模最大的數據集之一。Waymo 數據集共包含 1950 多個自動駕駛片

段，每一片段包含 20 秒的連續駕駛畫面。此資料集除了收集美國加州的鳳凰城、柯克蘭、山景城、舊金山等地區的市區或郊區，也包含了各種天氣狀況，例如晴天、雨天、白天或夜晚等。

在 Waymo 官方網站上的下載區，能夠下載 Waymo 完整的資料集，由於資料集之龐大，下載下來的格式為 tfrecord 壓縮檔，為一種 Tensorflow 儲存格式。其中，一個 tfrecord 檔案中約包含 199 幀 20 秒的連續畫面資料和所有 sensor 數據和相對應的 label 資料。在每一個 frame 的資料裡面分別包含五個不同角度的 images，分別為 FRONT、FRONT\_LEFT、FRONT\_RIGHT，前三種相片解析度為 1280×1920，SIDE\_LEFT、SIDE\_RIGHT，後兩種相片的解析度為 886×1920。

剛剛也有提到而每個場景包含了 200 frames，1 frame 有 5 個視角。資料集有 'TYPE\_UNKNOWN', 'TYPE\_VEHICLE', 'TYPE\_PEDESTRIAN', 'TYPE\_SIGN', 'TYPE\_CYCLIST' 共 5 種，而此次用到的 2D detection challenge 只要辨識出的只有 'TYPE\_VEHICLE'、'TYPE\_PEDESTRIAN'、'TYPE\_CYCLIST' 三種類別。

我們使用 **v1.2, March 2020 in WAYMO Open Dataset**. [1] (總共下載了 3 包的 tar file，訓練資料:14302 筆、驗證資料:1973 筆)。




<input type="checkbox"/>	名稱	大小	類型	建立時間 ?	儲存空間級別	上次修改日期	公開存取權
<input type="checkbox"/>	 training_0000.tar	22.7 GB	application/octet-stream	2020年3月...	Standard	2020年3月...	已隱藏值
<input type="checkbox"/>	 training_0001.tar	22.7 GB	application/octet-stream	2020年3月...	Standard	2020年3月...	已隱藏值
<input type="checkbox"/>	 training_0002.tar	22.8 GB	application/octet-stream	2020年3月...	Standard	2020年3月...	已隱藏值

Fig. 2 The Dataset we used

## 2.2 Data augmentation

我們 Fig. 3 在進行訓練時，發現對於標記 Cyclist 的資料其資料量相對於其他種類來的較少，甚至在某些 Section 當中更是連一張 Cyclist 的圖片都沒有，因此我們針對有 Cyclist 的圖片進行 data augmentation，要注意這邊使用的不是網路中自帶的 data augmentation，而是我們在處理 WAYMO Open Dataset 時，將具有 Cyclist 標籤的資料進行增量，主要是利用讀取 tfrecord 檔案時，若是判別到有 Cyclist 這個標籤出現，那就會將該圖片進行額外的轉換：

- 圖片調亮
- 圖片調暗
- 圖片水平翻轉
- 裁切 1/5 左半部分
- 裁切 1/5 右半部分
- 裁切 1/5 上半部分
- 裁切 1/5 下半部分

隨著圖片的轉換，bounding boxes 的座標也必須做相對應處理和修正。因此，我們

就可將 Cyclist 類別的圖像增為 8 倍，如 Fig 3 所示

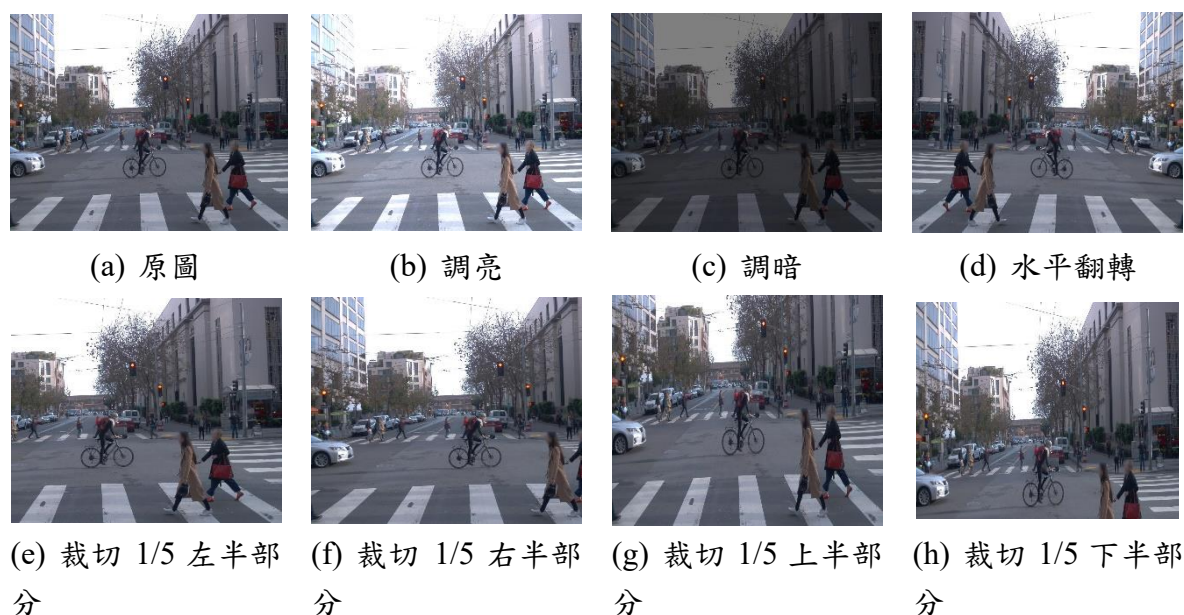


Fig. 3 影像增強示意圖

除了在讀取資料外的自己所做的 data augmentation 還有額外使用網路自身提供的 data augmentation，在 Yolo 的 cfg 中有提供 angle(角度), saturation(飽和度), exposure(曝光度), hue(色調)四種不同的方式來進行 data augmentation，在此次實驗中有嘗試調整 angle(角度)和 hue(色調)但有可能是由於本身在 Waymo Open Dataset 中就有包含了早晚光暗條件不同的照片，所以在 hue(色調)上比較沒感受到差別，但對於 angle(角度)以我們自己拍攝的突變來說就有較好的效果，推測是由於在排設過程中手部的晃動和不平行導致辨識效果有差異→Waymo Open Dataset 中資料都來至於架於車頂之相機相對平穩→如果在自己的裝置上建議額外生成一組資料來進行訓練。

```
batch=8
subdivisions=8
width=864
height=864
channels=3
momentum=0.9
decay=0.0005
angle=10
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=400
max_batches=2000
policy=steps
steps=3000
scales=.1
```

Fig. 4 Parameter for Yolo v3 Data Augmentation

## 2.3 Object detection

Object detection 目的在於定位和分類任何一張影像中可能存在的各種 objects，並且會用邊界框(bounding box)來顯示存在的可信度，在之前可能比較多人使用矩形的邊界框，雖然在框定邊界及計算上較為方便，但也在計算時浪費了更多的空間，所以有時會看到不規則形狀的邊界框更符合實際情況。Object detection 方法的框架主要可以分成兩類，其中一類是 Two-Stage，首先會產生區域的建議(region proposal)，再將每個建議做分類(classification)；另外一類則是 One-Stage，會將 object detection 視為迴歸(regression)或分類問題，直接採用統一的框架來達到最終的結果(類別與位置)。

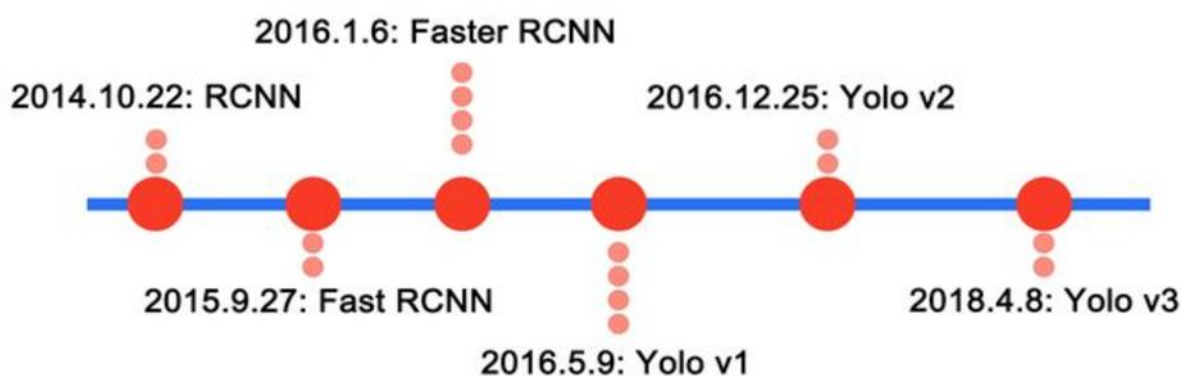


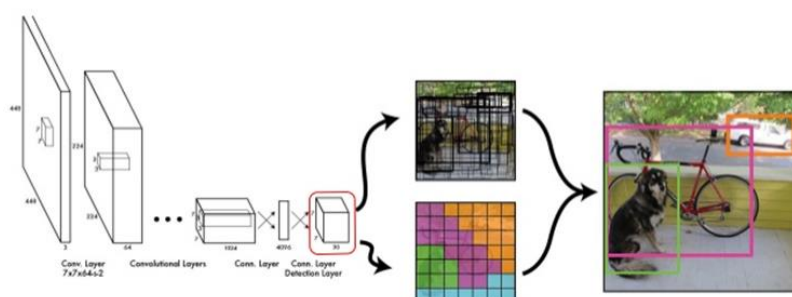
Fig. 5 Object detection 發展過程

### 2.3.1 One-Stage

為了解決 Two-Stage 為了提出 region proposal 導致拖垮速度的問題，有速度優勢的 One-Stage 模型開始出現，主要有兩大重要的框架：You Only Look Once (YOLO) [5] 和 Single Shot MultiBox Detector (SSD) [6]。

#### (a) Yolo

##### YOLO: You Only Look Once



Redmon et al. [You Only Look Once: Unified, Real-Time Object Detection](#), CVPR 2016

30

Fig. 6 Yolo(You Only Look Once)

YOLO 由 Redmon et al.於 2016 年在 CVPR 發表。YOLO 將 object detection 定義為一個迴歸問題，直接在圖像預測 bounding box (bbox)的座標、該 box 是否包含物件的信心度和物件的 class probabilities。

### 2.3.2 Two-Stage

Two-stage 的方法會先從整張圖片找出 Region Proposal (可能有包含物件的 Bounding Box)，再做迴歸分類。提出 Region Proposal 的方法很多，例如：Selective Search 和 RPN。Two Stage 的代表演算法有：R-CNN [2]，Fast R-CNN [3]，Faster R-CNN [4]。

#### (a) R-CNN

R-CNN 由 Ross Girshick 在 2014 提出，主要可以分成三個階段：Region proposal generation, CNN based deep feature extraction and Classification and localization. R-CNN 雖然當時對 PASCAL VOC 2012 的 mAP 上取得了大幅的進步，但它有幾個速度慢的問題：(1) 經由 Selective Search 提出的 Region Proposal 都要經過 CNN 做 feature extraction，過程相當耗時。(2) 從(1)中得到的 feature map 會變成 SVM 的訓練資料，因此並不是一個 end-to-end 的模型(SVM 的 loss 不會改動 feature map 的數值)。

#### (b) Fast-RCNN

Fast-RCNN 由 Girshick 於 2015 年所提出。在前半段與 RCNN 差別不大，同樣都透過 Selective Search 提出 Region Proposal，再將**整張圖**送到 CNN 提取特徵。提取特徵後，會先經過一個 RoI(Region of Interest) pooling layer，將不同 Region Proposals pooling 成一樣大小，再同時使用 softmax 和 bbox regressor 做分類和迴歸。

相較於 RCNN，原本每個 region proposal 都要經過 CNN，Fast-RCNN 只要將整張圖輸入，速度上取得很大的優勢。另外分類的部分改使用 Softmax 取代 SVM，並同時加入 BBOX regression 到模型中與分類一起收斂。但 Fast-RCNN 還是需要透過 Selective Search 提出 Region Proposal，沒有整合成 end-to-end。

#### (c) Faster-RCNN

Faster-RCNN 由 Ren et al.在 2015 年提出。最大的貢獻在使用了 Region Proposal Network (RPN)來提出 Region Proposal，取代了先前的 Selective Search，不僅在速度上又取得大幅進步，也達到真正的 end-to-end。但 Faster-RCNN 仍舊需要先提出 Region Proposal，整體速度還是受到限制。



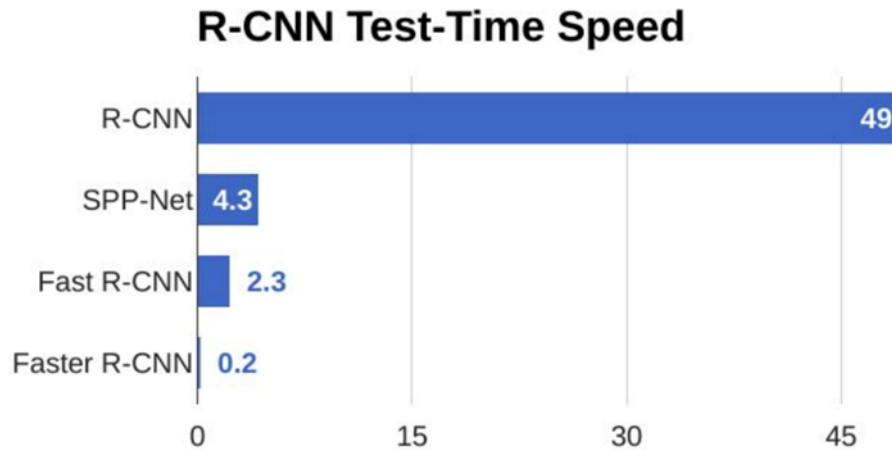


Fig. 7 R-CNN Test-Time Speed

## 3. Methods

### 3.1 YOLO

YOLO[7]系列(You Only Look Once, YOLO)是關於物件偵測(object detection)[8]-[10]的類神經網路演算法，可在單一照片上同時偵測多樣物件，除了預測各個物件可能性之外，還會框出預測物件的實際位置，此方法運用 convolutional neural network 的概念，以小眾架構 darknet[11]實作，沒有用到任何著名深度學習框架，輕量、依賴少、演算法高效率，就能達到物件偵測的效果。YOLO 目前已經有多種版本：

### 3.2 YOLO v1:

版本計算快速，能夠達到 real-time 速度需求，缺點是對位置的預測不夠精確，且小物體預測效果較差。

### 3.3 YOLO v2:

此版本對 v1 做了一些改善，首先引入 Faster-RCNN[4]中的 anchor box，不再直接取 mapping bounding box 的座標，而是預測相對於 anchor box 的參數，並使用 K-Means 求 anchor box 比例，並且去掉 fully connected layers，改成全部皆為 convolutional layer，每層加上 batch normalization layers，去掉 dropout layers。除此之外，更增加了 pretrain 解析度，從  $224 \times 224$  提升至  $448 \times 448$ 。

### 3.4 YOLO v3:

YOLO v3 並沒有做革命性的創新，而是參考其他的論文對本身的模型做優



化，效果十分顯著，主要分為兩種網路：

### 1. 使用 ResNet 網路(Residual Networks)

使用 Darknet-53[11]做為新的基底網路，採用了一般類神經網路加深時常用的 ResNet 結構來解決梯度問題，YoloV3 的 Backbone 從 Darknet-19 改為 Darknet-53 犧牲速度來提高精度，Darknet-53 用 53 層網路在 Imagenet 上面訓練，然後再增加 53 層訓練檢測任務，一共使用了 106 層卷積層，這是為什麼 YOLOv3 比 YOLOv2 要慢。

### 2. 使用 FPN 網路(Feature Pyramid Networks)

Neck 使用 FPN 多層級預測架構以提升小物體預測能力，特徵層從單層  $13 \times 13$  變成了多層  $13 \times 13$ 、 $26 \times 26$  和  $52 \times 52$ ，單層預測 5 種 bounding box 變成每層 3 種 bounding box (共 9 種)，詳見網路結構圖。使用 FPN 的架構可以讓低層較佳的目標位置和高層較佳的語義特徵融合，並且在不同特徵層獨立進行預測，使得小物體檢測改善效果十分明顯。

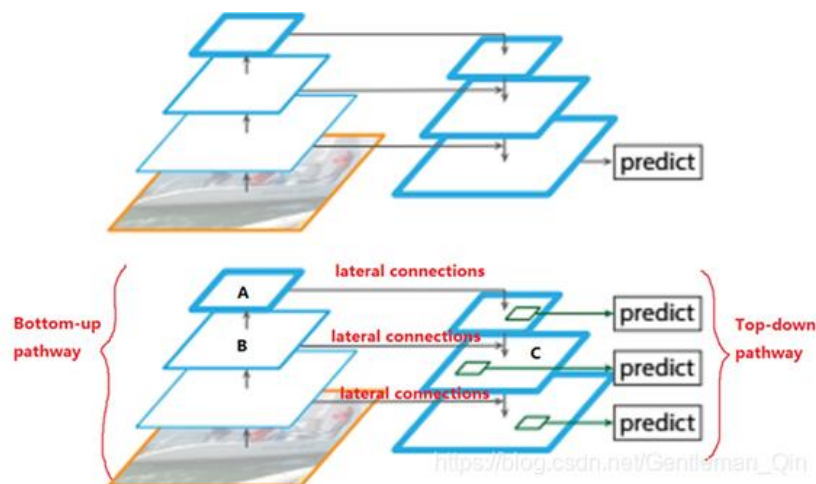


Fig. 8 FPN 架構圖[12]

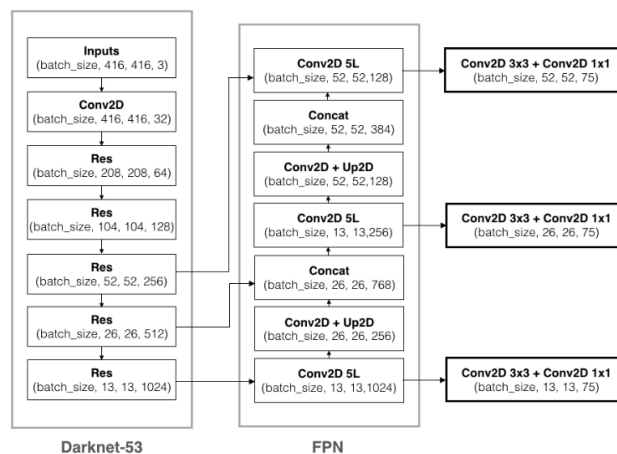
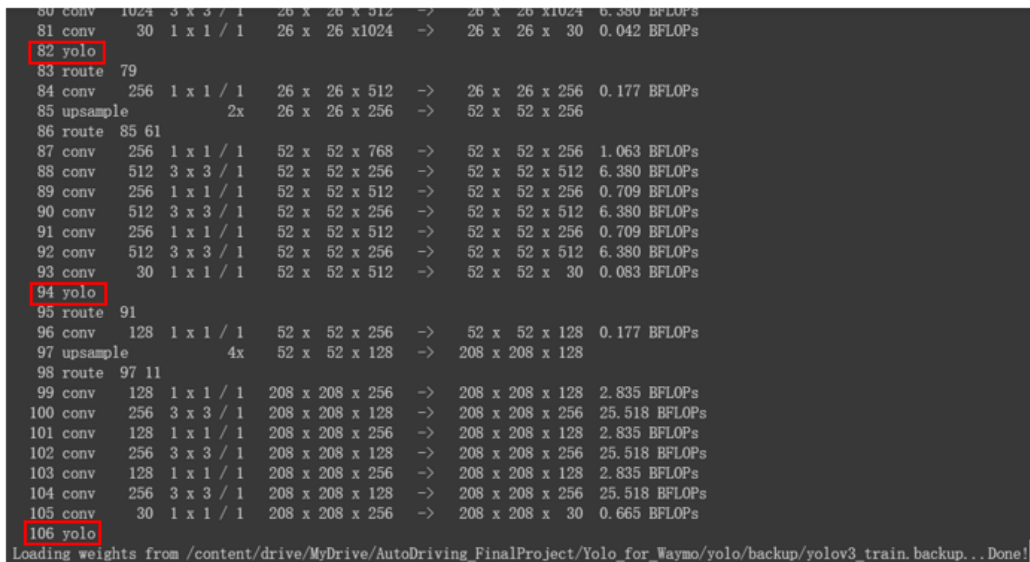


Fig. 9 YOLO v3 架構圖[7]

### YOLO v3 的模型架構：

YOLO 運用 convolutional neural network 的概念，而 YOLO v3 原本是以 53 層的 convolutional layers 組成，又被稱為 Darknet-53。對於偵測任務，又從原本的 Darknet-53 額外增加了 53 層的 layer，所以 YOLO v3 的模型架構總共會有 106 層的 layer，這也是為什麼我們在 load weights 的時候 output 出來的模型架構有 106 層，如下圖 Fig. 10。



Layer	Type	Filters	Kernel	Stride	Input Size	Output Size	BFLOPs
80	conv	1024	3 x 3	1	26 x 26 x 512	26 x 26 x 1024	6.380
81	conv	30	1 x 1	1	26 x 26 x 1024	26 x 26 x 30	0.042
82	yolo						
83	route	79					
84	conv	256	1 x 1	1	26 x 26 x 512	26 x 26 x 256	0.177
85	upsample		2x		26 x 26 x 256	52 x 52 x 256	
86	route	85 61					
87	conv	256	1 x 1	1	52 x 52 x 768	52 x 52 x 256	1.063
88	conv	512	3 x 3	1	52 x 52 x 256	52 x 52 x 512	6.380
89	conv	256	1 x 1	1	52 x 52 x 512	52 x 52 x 256	0.709
90	conv	512	3 x 3	1	52 x 52 x 256	52 x 52 x 512	6.380
91	conv	256	1 x 1	1	52 x 52 x 512	52 x 52 x 256	0.709
92	conv	512	3 x 3	1	52 x 52 x 256	52 x 52 x 512	6.380
93	conv	30	1 x 1	1	52 x 52 x 512	52 x 52 x 30	0.083
94	yolo						
95	route	91					
96	conv	128	1 x 1	1	52 x 52 x 256	52 x 52 x 128	0.177
97	upsample		4x		52 x 52 x 128	208 x 208 x 128	
98	route	97 11					
99	conv	128	1 x 1	1	208 x 208 x 256	208 x 208 x 128	2.835
100	conv	256	3 x 3	1	208 x 208 x 128	208 x 208 x 256	25.518
101	conv	128	1 x 1	1	208 x 208 x 256	208 x 208 x 128	2.835
102	conv	256	3 x 3	1	208 x 208 x 128	208 x 208 x 256	25.518
103	conv	128	1 x 1	1	208 x 208 x 256	208 x 208 x 128	2.835
104	conv	256	3 x 3	1	208 x 208 x 128	208 x 208 x 256	25.518
105	conv	30	1 x 1	1	208 x 208 x 256	208 x 208 x 30	0.665
106	yolo						

Loading weights from /content/drive/MyDrive/AutoDriving\_FinalProject/Yolo\_for\_Waymo/yolo/backup/yolov3\_train.backup... Done!

Fig. 40 Load weights 時的輸出模型 Layers

在這 106 層當中，會有 3 層用來偵測(detection)，分別是第 82 層、第 94 層和第 106 層，也就是在 Fig 11. 中用紅框標示出來的部分。

此架構包含了一些基本要素，包括 Residual Blocks、Skip connections 和 Up-sampling。

在每一個 CNN layer 後面都接著 Batch normalization layer 和 Leaky Relu 作為 activation function。

Model 並不會用到 pooling layers，而是採用額外的 convolutional layers，將其 stride 設為 2，用來把 feature map 做 down-sampling。因為這個額外的 convolutional layers 可以用來預防 low-level features 的 loss，這樣一來就能增加小物件被偵測到的可能性，然而 pooling layer 無法做到這件事。

提一個簡單的範例，從 Fig. 11 中可看出，如果原本的 feature map 在經過 max pooling 之後，只會留下該區塊中的最大值，而若是經過 stride 為 2 的 convolutional layer，會留下該區塊的計算總和，這樣平均的概念，比較能表示該特徵的特性，而不是該特徵的極端值。

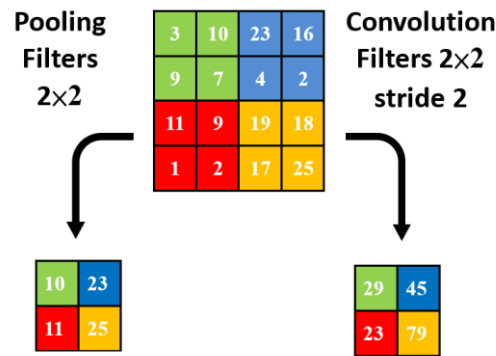


Fig. 11 Pooling layer 和 Convolutional layer 比較[7]

### YOLO v3 的輸入：

其輸入為一批訓練圖片，shape 為  $(n, 416, 416, 3)$ ， $n$  代表是訓練圖片的總數，而 416 為 network 圖像的寬度和高度，3 所代表的是 RGB 總共三個 channel。其實這個圖像的寬度和高度  $(416 \times 416)$  並非代表我們丟入的圖片需要是這個大小，而是我們將圖片丟進來後都會先 resize 成這個大小再來進行訓練。並且可以透過 Darknet, Tensorflow, Keras 等架構調正參數來決定是否要保留圖像的長寬比例，比較兩種方式的結果，就可找到最適合自己資料集的方式。

Network 圖像的寬度和高度是可以調整的，當然，數入圖像的解析度愈高，訓練過後的結果的準確性就會愈高，只要是可以整除 32 的數就可作為寬度和高度，原因會再後面的部分再作介紹。

### YOLO v3 的偵測：

YOLO v3 在三種 scales 和三個 layers 下進行偵測，三種 scales 為 network 將數入圖像做 down-sampling 的三個 factors 分別是 32、16 和 8，而三個 layers 是剛剛前面有提到模型中的第 82 層、第 94 層和第 106 層。

假設現在我們的 network 輸入圖像 shape 是  $(416, 416)$ ，如果經過 down-sampling scale 為 32，那我們會得到  $13 \times 13$  大小的圖像，那這會拿來偵測大物件，如果 scale 為 8，則得到  $52 \times 52$  大小圖像，則可用來偵測小物件。這也就是為什麼在前面的時候我們說 network 的輸入寬度和高度必須能被 32 整除，這樣才能進行 down-sampling 的過程，能被 32 整除了，那也同樣能被 16 和 8 整除。

### YOLO v3 的 detection kernels：

有了前面提到的 down-sampling 圖像，在經過我們  $1 \times 1$  的 kernel 後，大小仍會保持不變，也就是說  $13 \times 13$  的 down-sampling 圖像仍然保持  $13 \times 13$  的大小，而  $52 \times 52$  的 down-sampling 圖像也仍然保持  $52 \times 52$  的大小，只是我們的 kernel 有 depth，他的 depth 是由下面這個式子計算而得：

$$depth = b * (5 + c)$$

b 所代表的是在 feature map 的每個 cell 下可以預測的 bounding boxes 個數，而 c 所代表的是 class 的數目，也就是要辨識的種類數目。YOLO v3 會在每個 cell 預測 3 個 bounding boxes，所以我們可以得知  $b = 3$ ，而每個 bounding boxes 會有  $5 + c$  個屬性，前面 5 個屬性分別是 center of x、center of y、width of box、height of box 和 objectness score，而那額外加的 c 個屬性就分別是那 c 個 classes 的 confidence 值，像我們的 Waymo 資料集總共有 5 個 classes，所以我們就須將 config 檔中的一些參數更正如下 Fig. 13 所框出來的部分。

```

601 [convolutional]
602 size=1
603 stride=1
604 pad=1
605 # filters = (num/3) * (5+classes)
606 filters=30
607 activation=linear
608
609 [yolo]
610 mask = 6,7,8
611 anchors = 10, 13, 16, 30, 33, 23, 30, 61, 62, 45, 59, 119, 116, 90, 156, 198, 373, 326
612 classes=5
613 num=9
614 jitter=.3
615 ignore_thresh = .5
616 truth_thresh = 1
617 random=1

```

Fig. 12 YOLO v3.cfg 調整

經過 kernel 後 feature maps 的 shape 就會變為 (13, 13, 30)、(26, 26, 30)和(52, 52, 30)。

### YOLO v3 的 grid cell：

1. 一個物件會有一個 ground truth bounding box。
2. 其 center cell 會被 YOLO v3 用來預測這個物件。
3. 這個 cell 的 objectness score 會被設為 1。

在訓練所有 cell 的時候，每個 cell 都會產生三個 predict bounding boxes，那我們又該選擇哪一個？又如何選出最好符合該物件的 bounding boxes，那接下來我們會用 anchor boxes 來介紹。

### YOLO v3 的 anchor boxes：

Anchor boxes 為 YOLO v3 的 pre-defined default boxes，也可稱作 priors。那這些 bounding boxes 是用來計算預測的 bounding boxes 的真正位置和寬度和高度，在 YOLO v3 中總共用了 9 個 anchor boxes，如 Fig. 14 所示，每個 scale 有 3 種 anchor boxes，這代表在每個 scale 下的每個 cell 都會由 3 種 anchor boxes predict 出 3 個 bounding boxes，所以 scale 32 會產生  $13 \times 13 \times 3 =$

507 個 bounding boxes，scale 16 會產生  $26 \times 26 \times 3 = 2028$  個 bounding boxes，scale 8 會產生  $52 \times 52 \times 3 = 8112$  個 bounding boxes，所以 YOLO v3 總共需要 predict 10647 個 bounding boxes 在一張圖片上，那這些 anchor boxes 的計算方式就是由 k-means clustering 而得。

611 anchors = 10, 13, 16, 30, 33, 23, 30, 61, 62, 45, 59, 119, 116, 90, 156, 198, 373, 326

Fig. 13 The offset of YOLO v3 anchor boxes

### YOLO v3 predict bounding boxes :

首先計算相對於 anchor 的 offset，這也被稱作 log-space transform，YOLO v3 用 sigmoid function 作為 activation function 得到輸出，下方為取得 predicted bounding box 的計算方式：

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w * e^{t_w}$$

$$b_h = p_h * e^{t_h}$$

其中  $b_x, b_y, b_w, b_h$  是 predicted bounding box 的位置、寬度和高度

$t_x, t_y, t_w, t_h$  是 Network 的 outputs

$c_x, c_y$  是 anchor box 的左上角位置

$p_w, p_h$  是 anchor box 的寬度和高度

$\sigma()$  為 sigmoid function，其值會界於 0 到 1 之間

YOLO v3 並非採用絕對大小，而是採用相對大小來預測相對於 anchor box 的 offset，這樣一來就可以在訓練的時候消除 unstable gradients，那就是為甚麼  $c_x, c_y, p_w, p_h$  會先除以 image width 和 image height 來做 normalize。

所以再將  $b_x, b_y, b_w, b_h$  乘以 image width 和 image height 就可得到真正的位置和大小：

$$BB_x = b_x * width\ of\ image$$

$$BB_y = b_y * height\ of\ image$$

$$BB_w = b_w * width\ of\ image$$

$$BB_h = b_h * height\ of\ image$$

### YOLO v3 objectness score :

Objectness score 是 bounding box 的其中一種屬性，若是 ground truth bounding box 中愈靠近 center cell 的 objectness score 就愈接近 1，而愈靠近 corner cell 的 objectness score 就愈接近 0。而這個數字即代表了預測是物件的 center cell 的可能性，這和 confidence list 不一樣的是 confidence list 是代表這個 object 是車或是行人或是騎行者的分別可能性，而 objectness score 所代表的是該 bounding box 圈出來的會是物件的可能性。

Objectness score ( $p_0$ ) 的計算公式：

$$P_{object} * IoU = \sigma(t_0) = p_0$$

$P_{object}$  為 predicted probability

$IoU$  為 ground truth bounding box 和 predicted bounding box 之間的 intersection over union :

$$IoU = \frac{BB_1 \cap BB_2}{BB_1 \cup BB_2}$$

### 3.5 YOLO v4:

YOLO v4 並非 YOLO 系列的原作者 Joseph Redmon 所開發，而是由 AlexeyAB 繼承了 YOLO 系列的理念和思想，在 YOLOv3 的基礎上不斷進行改進和開發，於 2020 年 4 月發布，同時獲得了原作者 Joseph Redmon 的承認。YOLOv4 可以使用傳統的 GPU 進行訓練和測試，並能夠獲得實時的，高精度的檢測結果。

#### YOLO v4 的模型架構：

YOLOv4 採用的模型架構為 CSPDarknet53，為上一代 v3 的模型架構 Darknet53 加上 CSPNet (Cross Stage Parital Network) 所組成。CSPNet 主要從網絡結構設計的角度解決推理中計算量大的問題。CSPNet 的作者認為推理計算過高的問題是由於網絡優化中的梯度信息重複導致的。因此採用 Cross-Stage-Partial-connections (CSP) 先將基礎層的特徵映射劃分為兩部分，然後通過跨階段層次結構將它們合併，在減少了計算量的同時可以保證準確率。

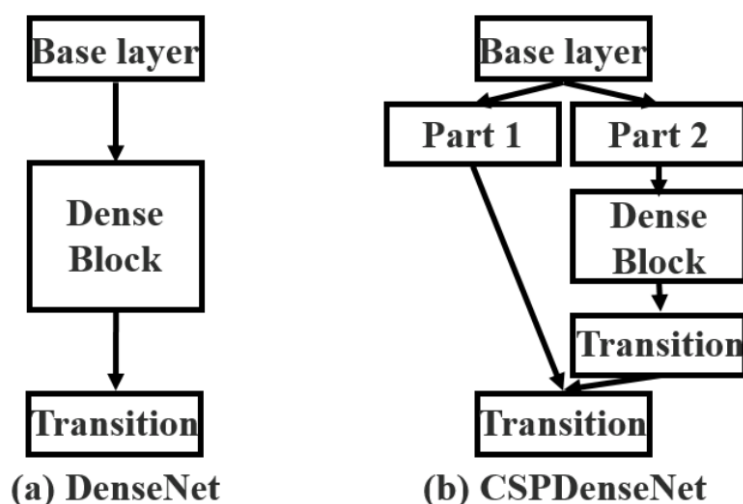


Fig. 14 DenseNet 架構圖[13]

CSPNet 可以在不降低甚至增加準確度的情況下，減少 CNN 網路 10%到 20% 的計算量。Yolov4 在主幹網絡 Backbone 採用 CSPDarknet53 網絡結構，主要有三個方面的優點：



優點一：增強 CNN 的學習能力，使得在輕量化的同時保持準確性。

優點二：降低計算瓶頸。

優點三：降低內存成本。

### SPP (spatial pyramid pooling layer)：

在 YOLOv4 中，對 SPP 進行了修改以保證輸出為空間維度。最大池化的核大小為  $k = \{1 \times 1, 5 \times 5, 9 \times 9, 13 \times 13\}$ 。將來自不同核大小池化後的特徵圖串聯在一起作為輸出。採用 SPP 層的方式，比單純的使用單個尺寸核大小的最大池化的方式，更有效的增加主幹網絡的感受野，顯著的分離最重要的上下文特徵。

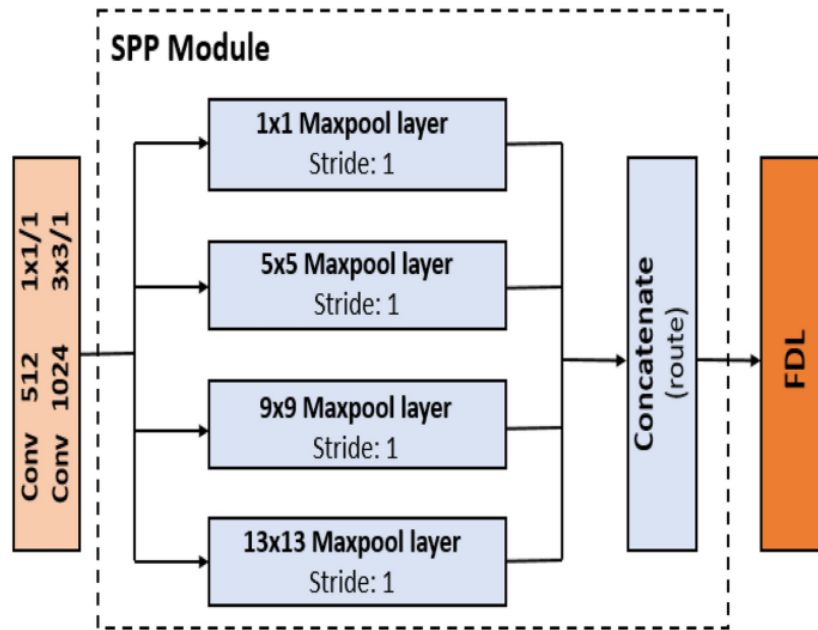


Fig. 15 SPP 架構圖[14]

### PAN (Path Aggregation Network)：

FPN 是自頂向下的，將高層的特徵信息通過上採樣的方式進行傳遞融合，得到進行預測的特徵圖。

Yolov4 中 Neck 這部分除了使用 FPN 外，還在此基礎上使用了 PAN 結構：和 Yolov3 的 FPN 層不同，Yolov4 在 FPN 層的後面還添加了一個自底向上的特徵金字塔。

這樣結合操作，FPN 層自頂向下傳達強語義特徵，而特徵金字塔則自底向上傳達強定位特徵，兩兩聯手，從不同的主幹層對不同的檢測層進行參數聚合。

值得注意的是，一般的 PAN 對相鄰的特徵圖是使用相加的方法，而在 YOLOv4 作者的採用中，則是改成 concatenation 的方式。



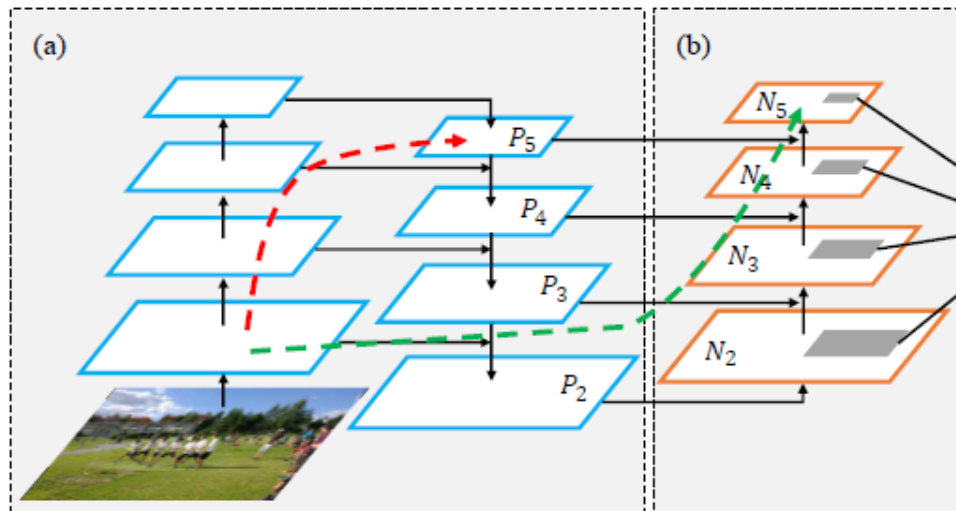


Fig. 16 PAN 示意圖[15]

### Mosaic 數據增強：

YOLOv4 中有使用許多不同的優化策略，其中 Mosaic 是一種數據增強方法，將 4 張訓練圖像組合成一張進行訓練（而不是 CutMix 中的 2 張），這增強了對超出正常圖像邊框的目標的檢測，增加樣本多樣性。另外，每個 mini-batch 包含大量的圖像（是原來 mini-batch 所包含圖像數量的 4 倍），因此，在估計均值和方差時減少了對 large mini-batch sizes 的需求。

### CIOU\_loss：

目標檢測任務的損失函數一般由 Classification Loss(分類損失函數)和 Bounding Box Regression Loss(回歸損失函數)兩部分構成。在 YOLOv3 中，採用的是 IOU\_Loss，而 YOLOv4 則使用 CIOU\_loss。IOU\_Loss 主要考慮檢測框和目標框重疊面積，在 IOU 的基礎上，GIOU\_Loss 解決邊界框不重合時的問題。DIOU\_Loss 在 IOU 和 GIOU 的基礎上，進一步考慮邊界框中心點距離的信息。CIOU\_Loss 就是在 DIOU 的基礎上，多考慮了邊界框寬高比的尺度信息。Yolov4 中採用了 CIOU\_Loss 的回歸方式，使得預測框回歸的速度和精度更高一些。

## 3.6 YOLO v5:

YOLO v5 在 YOLO v4 發表不久即登場，與 YOLO v3 以及 YOLO v4 的作者皆不同，作者為 Glenn Jocher。YOLO v5 因為作者遲遲沒有發表論文，並且相比於 YOLO v4 的創新性也不足，因此對於能否稱為 YOLO v5 頗有爭議，不過 YOLO v5 中仍有許多地方值得研究探討，整體來說，也是不錯的類神經網路演算法，值得深入。

### YOLO v5 的模型架構：

Yolov5 一共有四種網絡模型，分別為 Yolov5s、Yolov5m、Yolov5l、Yolov5x。其中 Yolov5s 網絡是 Yolov5 系列中深度最小，特徵圖的寬度最小的網絡。後面的

3 種都是在此基礎上不斷加深，不斷加寬。YOLOv5 採用的模型架構為 New CSP-Darknet53，在 YOLOv4 中，也使用了 CSPNet 的設計思路，在主幹網絡中設計了 CSP 結構，而 YOLOv5 除了在主幹網絡使用了 CSP 結構之外，Neck 中使用了 New CSP-PAN 的結構，以 YOLOv5s 網絡為例，CSP1\_X 結構應用於 Backbone 主幹網絡，另一種 CSP2\_X 結構則應用於 Neck 中。

### SPPF (spatial pyramid pooling fast layer)：

Yolov5 使用的 SPPF 與前系列的 SPP 之間最大的差異比較，在於 SPP 以  $k = \{1 \times 1, 5 \times 5, 9 \times 9, 13 \times 13\}$  的最大池化層並行處理並進行 concat，而 SPPF 則是以三個  $k = 5 \times 5$  的最大池化層以串行的方式依須處理，經過兩個  $k = 5 \times 5$  的最大池化層的輸出，其結果等同於經過  $k = 9 \times 9$  的輸出；經過三個  $k = 5 \times 5$  的最大池化層的輸出，其結果等同於經過  $k = 13 \times 13$  的輸出，由於  $k = 5 \times 5$  處理所需花費的時間顯著少於  $k = 13 \times 13$  處理所需花費的時間，因此採用 SPPF 的方式相比 SPP 能夠提升更高的效率。

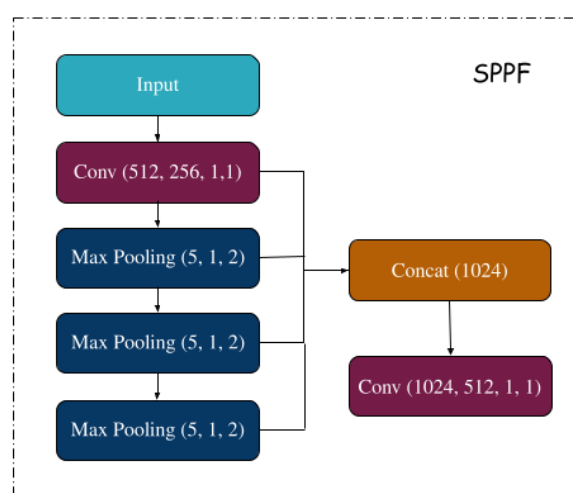


Fig. 17 SPPF 示意圖[16]

### 數據增強：

Yolov5 中使用了許多數據增強的策略，除了 YOLOv4 中有使用過的 Mosaic 之外，以下列出一些其他的方法：

- ◆ Copy paste：將圖片部件進行複製並黏貼到新的圖上，以增加樣本多樣性。
- ◆ Random affine：作者有對四個要點進行隨機仿射(Rotation, Scale, Translation and Shear)，不過初始設定只有 Scale, Translation 設為 1，其餘兩個為 0，若有需要 Rotation 和 Shear，須另行設定。
- ◆ MixUp：將兩張圖片按照一定透明程度混合圖片，在 YOLOv5 中，有 10% 的機會會啟用這功能。
- ◆ Albumentations：默認的情況下下載 YOLOv5 並不會安裝此功能，擁有濾

波、直方圖均衡化以及改變圖片質量等功能。

- ◆ Augment HSV : 對圖片的 Hue,Saturation,Value 變化以增加樣本數。
- ◆ Random horizontal flip : 將圖片按照一定比例隨者水平方向進行翻轉。

### 訓練策略：

除了數據增強之外，YOLOv5 也有許多的訓練策略，以下列出裡面有使用的方法：

- ◆ Multi-scale training(0.5~1.5x):將原始輸入圖片以 0.5 倍至 1.5 倍的比例進行放大縮小，來進行多尺度訓練，取 32x32 的整數倍。
- ◆ Auto Anchor(For training custom data):如果數據集的目標或者長寬等比例跟一般常見的差異要大的話，可以自動生成適合的 anchor，適合用在需要客製化訓練資料的時候。
- ◆ Warmup and Cosine LR scheduler:Warmup 為在訓練初期會將學習率從一個非常小的值慢慢增長到我們所設置的初始學習率，相當於一個熱身訓練；而 Cosine LR scheduler 則是以 cosine 的變化形式慢慢降低學習率。
- ◆ EMA(Exponential Moving Average): 相當於給學習變量加上一個動量，這樣在更新參數的時後就會更加的平滑一些。
- ◆ Mixed precision:混合精度訓練，啟用能減少 GPU 的顯存的佔用，理論上能夠減半，這對於訓練非常大的模型是非常有用的，其次還能加快網絡的訓練，理想狀況下加速兩倍。

### 損失計算：

如(1)所示，YOLOv5 的損失計算可以分成三大類的加總，分別為 Classes loss, Objectness loss ,Location loss。第一個分類損失，採用的是 BCE loss,注意只計算正樣本的分類損失。第二個 obj 損失，採用的是 BCE loss,注意這裡的 obj 指的是網絡預測的目標邊界框與 GT Box 的 CIoU。這裡計算的是所有樣本的 obj 損失。第三個定位損失，採用的是 CIoU loss,注意只計算正樣本的定位損失。

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc} \quad (1)$$

其中 $\lambda_1$ 、 $\lambda_2$ 、 $\lambda_3$ 為平衡參數。

### 平衡不同尺度損失：

針對三個預測特徵層(P3,P4,P5)上的 obj 損失(2)採用不同的權重，P3 是針對小型目標的一個預測特徵層，大小為  $80 \times 80 \times 256$ ，P4 針對的是一個中等大小目標的預測特徵層，大小為  $40 \times 40 \times 512$ ，P5 對應的就是大目標的預測特徵層，大小為  $20 \times 20 \times 1024$ 。小型目標的權重值為 4.0，中型目標的權重值為 1.0，大型目標的權重值為 0.4，由於小型目標更難預測一些，因此權重要大一些。

$$L_{obj} = 4.0 \cdot L_{obj}^{small} + 1.0 \cdot L_{obj}^{medium} + 0.4 \cdot L_{obj}^{large} \quad (2)$$

## 4. Result(YOLO 實作)

### 4.1 YOLO v3 vs. YOLO v4

- Cfg 檔變數建置

```
[net]
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
learning_rate=0.0013
burn_in=1000
max_batches = 6000
policy=steps
steps=4800,5400
scales=.1,.1
```

## ● 訓練次數

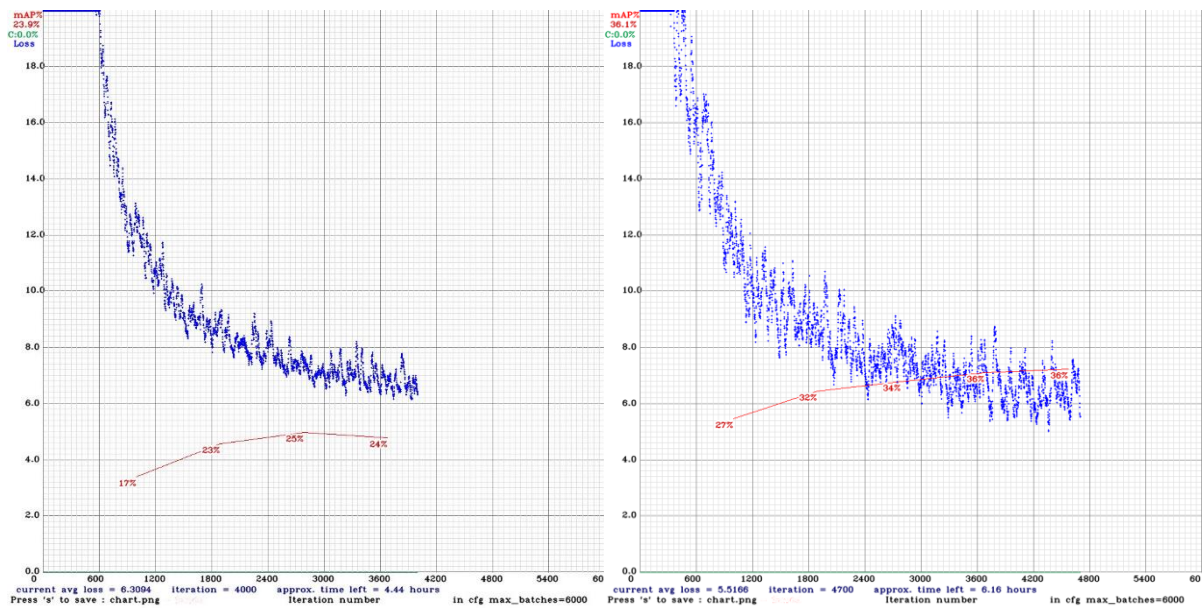


Fig. 18 Comparison of Yolo v3 and Yolo v4 (left: v3, right: v4)

圖片辨識結果採 1000、3000、4000 Iterations 做比較，因為從 loss 圖(19)中可看見在訓練次數達 3000 時，loss 趨近於平緩，因此往前取一個 weight 做圖片辨識的比較，往後也取一個 weight 做比較。

## ● 圖片辨識結果比較



Fig. 19 Yolov3 with 1000 iterations

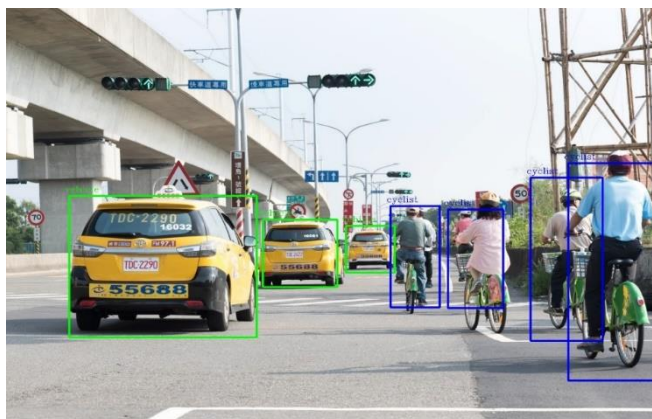


Fig. 20 YOLOv4 with 1000 iterations

從以上結果可發現，YOLOv4 已能初步框選出正確的物件，但精度還不佳，舉中間奶奶騎的腳踏車 label 來說，幾乎沒對到，而 YOLOv3 更不用說，框選的能力極差



Fig. 21 YOLOv3 with 3000 iterations

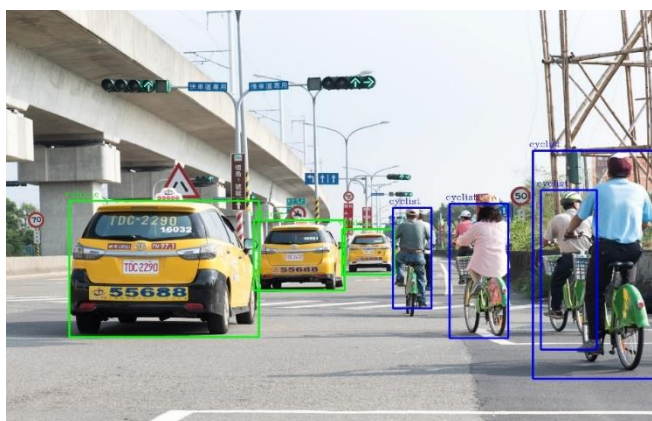


Fig. 22 YOLOv4 with 3000 iterations

從以上結果可發現，YOLOv4 已能框選出正確的物件，但準確度還可以再精修，舉最右側阿伯的腳踏車來說，label 應該可以框的較窄一些，而 YOLOv3 框選的能力則提升許多，然而精度依舊不佳。



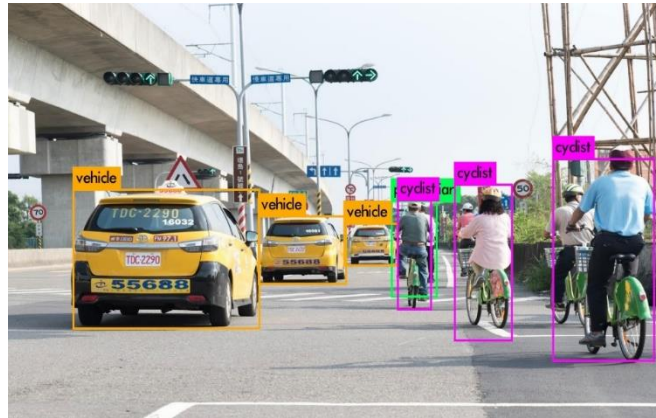


Fig. 23 YOLOv3 with 4000 iterations

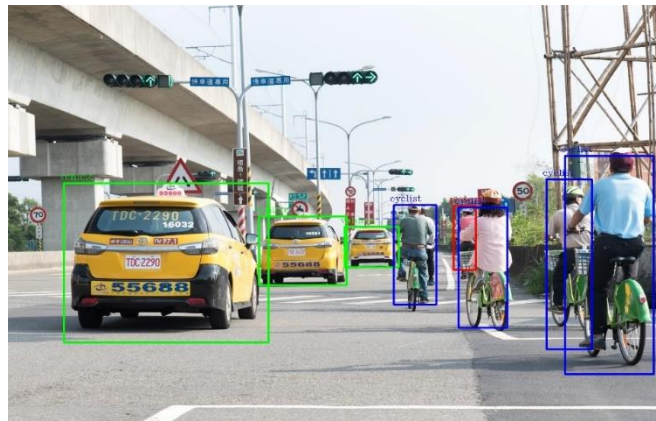


Fig. 24 YOLOv4 with 4000 iterations

從以上結果可發現，YOLOv4 最終不僅能框選出正確的物件，框的位置也很精準，而 YOLOv3 框選的能力則約等於 YOLOv4 訓練 3000 次的結果。綜上所述，YOLOv4 的辨識結果明顯優於 YOLOv3，雖然花費的時間相對 YOLOv3 來的久(前者訓練 1000 Iterations 約 4 小時，後者約 2 小時)，其框選的能力也高出許多。



## ● Map、recall、f1score 辨識結果比較

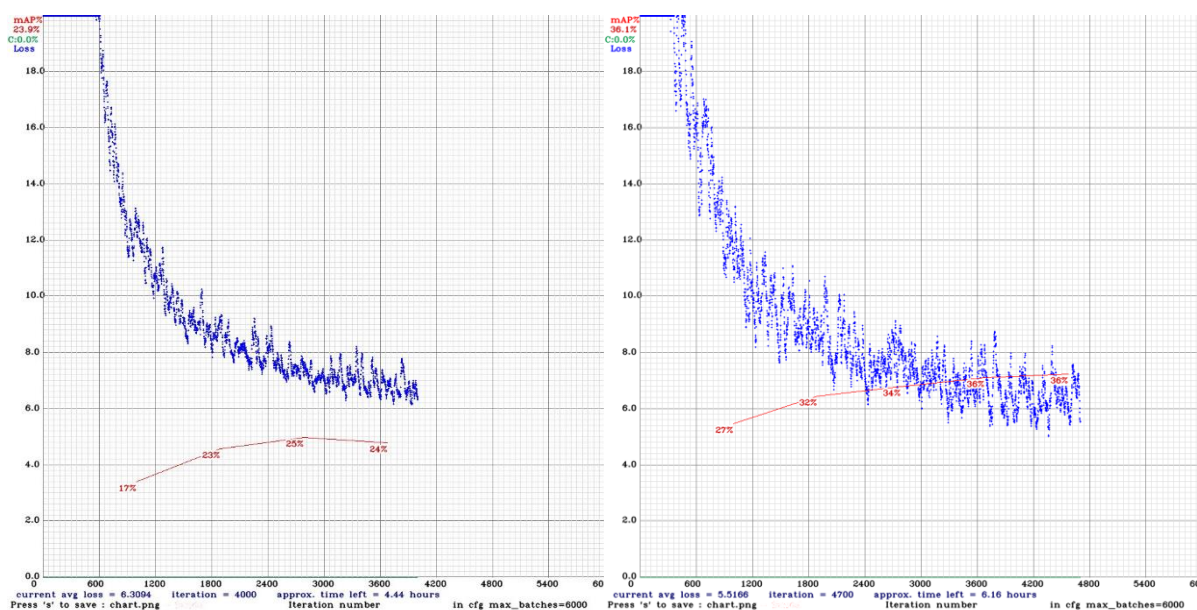


Fig. 25 Map Comparison of Yolo v3 and Yolo v4 (left: v3, right: v4)

```
detections_count = 101529, unique_truth_count = 28834
class_id = 0, name = cyclist, ap = 0.00% (TP = 0, FP = 2)
class_id = 1, name = vehicle, ap = 53.72% (TP = 15340, FP = 8677)
class_id = 2, name = pedestrian, ap = 20.90% (TP = 679, FP = 1101)

for conf_thresh = 0.25, precision = 0.62, recall = 0.56, F1-score = 0.59
for conf_thresh = 0.25, TP = 16019, FP = 9780, FN = 12815, average IoU = 46.62 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.248737, or 24.87 %
```

Fig. 26 Yolo v3 舊訓練結果

```
detections_count = 101332, unique_truth_count = 28834
class_id = 0, name = cyclist, ap = 0.00% (TP = 0, FP = 1)
class_id = 1, name = vehicle, ap = 66.27% (TP = 17743, FP = 7113)
class_id = 2, name = pedestrian, ap = 39.95% (TP = 904, FP = 1162)

for conf_thresh = 0.25, precision = 0.69, recall = 0.65, F1-score = 0.67
for conf_thresh = 0.25, TP = 18647, FP = 8276, FN = 10187, average IoU = 54.21 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.354076, or 35.41 %
```

Fig. 27 Yolo v4 舊訓練成果

從以上圖片可發現，cyclist 種類的 ap 結果為零，這導致了兩者的 map 極低，但經過影片實作，呈現的結果 cyclist 框選能力卻很好，使我們懷疑 validation data

是否出了問題。經檢查後發現，隨機取樣的照片並無 cyclist 種類，這說明以上的訓練成果並非正確的，因此我們又另外處理了另一組 validation data 來做測試，成果如下：

```
detections_count = 1217308, unique_truth_count = 402342
class_id = 0, name = cyclist, ap = 52.89%      (TP = 2219, FP = 1907)
class_id = 1, name = vehicle, ap = 62.85%      (TP = 186907, FP = 75337)
class_id = 2, name = pedestrian, ap = 51.99%   (TP = 62693, FP = 40889)

for conf_thresh = 0.25, precision = 0.68, recall = 0.63, F1-score = 0.65
for conf_thresh = 0.25, TP = 251819, FP = 118133, FN = 150523, average IoU = 50.80 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.559091, or 55.91 %
```

Fig. 28 Yolo v3 新訓練成果

```
detections_count = 1154637, unique_truth_count = 402342
class_id = 0, name = cyclist, ap = 68.06%      (TP = 2819, FP = 1582)
class_id = 1, name = vehicle, ap = 75.58%      (TP = 216696, FP = 60303)
class_id = 2, name = pedestrian, ap = 68.96%   (TP = 73561, FP = 31455)

for conf_thresh = 0.25, precision = 0.76, recall = 0.73, F1-score = 0.74
for conf_thresh = 0.25, TP = 293076, FP = 93340, FN = 109266, average IoU = 59.46 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.708673, or 70.87 %
```

Fig. 29 Yolo v4 新訓練成果

由以上成果可得知，同樣是 3000 iterations 的情況下，Yolov3 的 MAP=55.91%、recall=0.63、F1-score = 0.65；Yolov4 的 MAP = 70.87%、recall=0.73、F1-score = 0.74，所有評估結果 Yolov4 都較 Yolov3 來的好，因此影片辨識成果以 Yolov4 去呈現。

## ● Yolov4 影片辨識結果

訓練的環境是以 Colab 去做進行，而 Colab 不能有彈出式視窗，因此不管在執行訓練、圖片辨識、影片辨識時皆需要在指令尾部加上 `-dont_show`，不然舉影片辨識為例，就會發生以下報錯情形：

```
Unable to init server: Could not connect: Connection refused
(Demo:7835): Gtk-WARNING **: 12:01:08.615: cannot open display:
```

Fig. 30 Colab 報錯

從影片結果發現，做 Data augmentation 將 Cyclist 資料變多，使得影片辨識腳踏車的效果非常顯著，舉 Yolov4 test1 影片中的片段為例，如下圖：

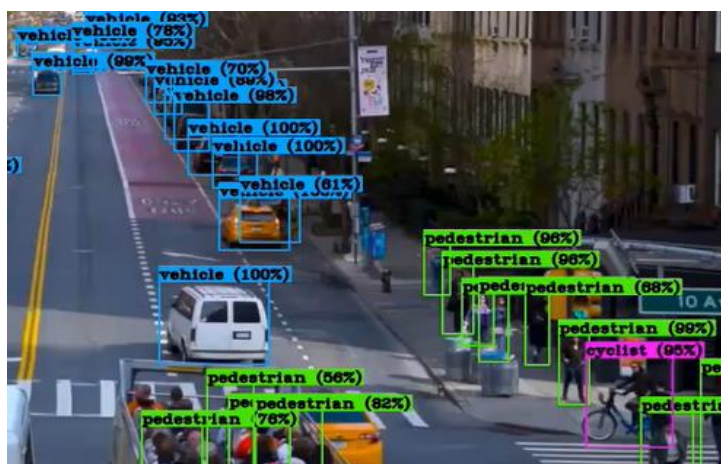


Fig. 31 YOLOv4 test1 有腳踏車片段

可以發現在多數行人中穿越的腳踏車，辨識過程依舊能穩穩地標起來，這說明對資料的處理對成果發揮了很大的作用。但在沒有設閾值的影片中，有時依舊會誤框錯誤的物件，舉 YOLOv4 test1 下方片段來說，左側圖片會將街道邊的樹框成行人，而在設下閾值 0.5 後，就能減少影片中誤判的情形發生。

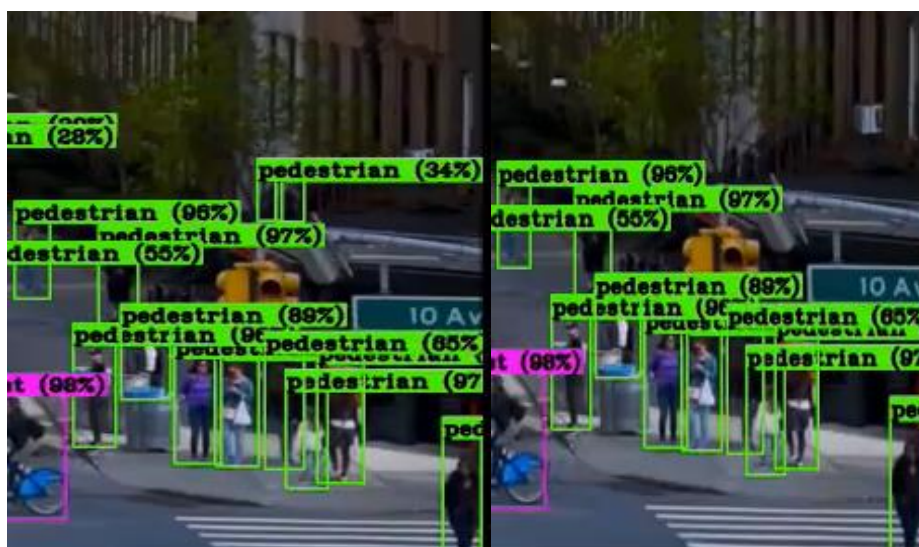


Fig. 32 YOLOv4 test1 有無法閾值比較 (左有, 右無)

再舉 YOLOv4 test4 下方片段來說，左側圖片會將彎路旁的小路面框成車輛，而在設下閾值 0.5 後，能除去誤判的情形發生。



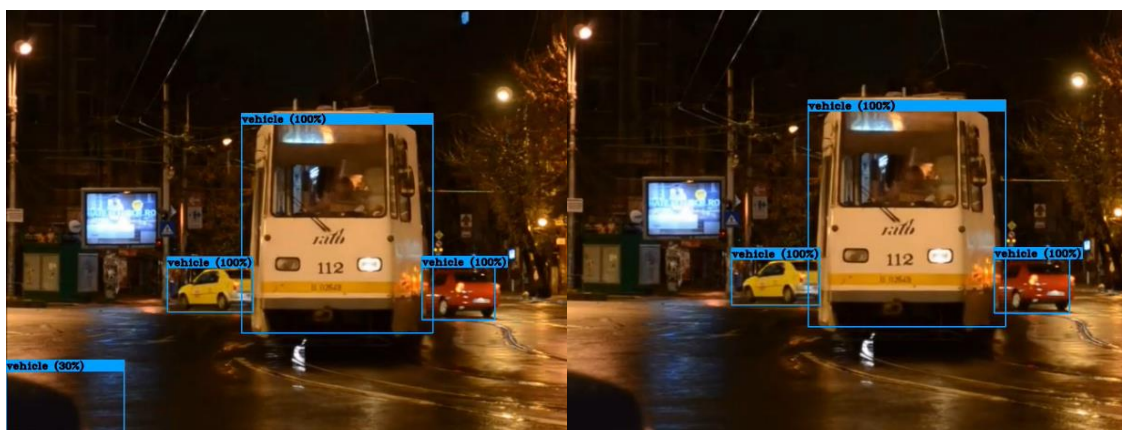


Fig. 33 YOLOv4 test4 有無法閾值比較 (左有，右無)

影片結果的呈現我們也有做篩選，test1 與 test2 是在白天辨識的影片，test3 與 test4 則是在晚上辨識的影片，由於資料集中擁有不同時段的照片，因此辨識結果在白天或黑夜皆能有很好的辨識成果。

下圖是 YOLOv4 test2 的片段，這段影片選用白天滿街車輛的影片做測試，而測試結果可以發現每台車幾乎都能精準地框選出來，不僅如此，連街道兩側行人也能辨識的到，右方行人若是不細看，連人眼都未必能發現，因為兩側的樹木會遮蔽行人。

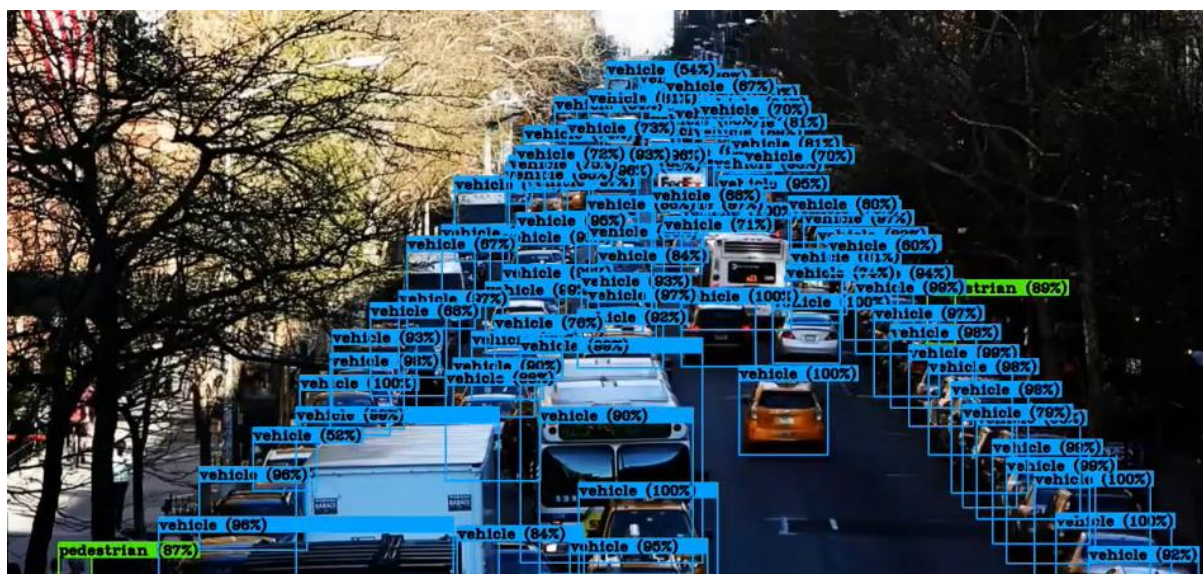


Fig. 34 YOLOv4 test2 白天滿街車輛

最後，是 YOLOv4 test3 的片段，這段影片選用黑夜車輛快速行駛的影片來做辨識，而 YOLOv4 辨識的成果也不錯。



Fig. 35 YOLOv4 test3 黑夜車輛快速行駛

綜合以上結果，可以說我們成功訓練出能克服各式情況的模型權重，不管白天黑夜、物件多寡、物件重疊程度、物件移動速度，這都歸功於資料前處理的重要性與多樣性。

以下是 YOLOv4 訓練影片連結：

1. [YOLOv4 test1](#)
2. [YOLOv4 test2](#)
3. [YOLOv4 test3](#)
4. [YOLOv4 test4](#)

這邊不做跟 YOLOv5 做比較的原因有兩點，第一是資料量不同，YOLOv5 在訓練的過程中因為一些原因有降低資料量，這在後續會進一步的討論，第二點是訓練次數計算方式不同，YOLOv3、YOLOv4 在訓練時以 Iterations 做計算，YOLOv5 則以 epochs 做計算

，因此難以在同個訓練程度下做比較，基於以上兩點，才將 YOLOv5 分開討論。

## 4.2 YOLO v5

### ● 環境建置

```
[ ] !git clone https://github.com/ultralytics/yolov5 # clone
    %cd yolov5
    %pip install -qr requirements.txt # install
```

Fig. 36 YOLOv5 環境建置

Yolov5 相較前幾代 Yolo，擁有更加完善的包裝功能以及容易部署環境的特色，訓練的步驟及輸出結果作者大都定義好了，因此使用上十分方便。本組使用的實作環境是 Google Colab，在環境建置上只需要 3 行程式碼

## ● 訓練資料及標籤

Yolov5 所需圖片格式為.jpg 檔及標籤資料為.txt 檔，但較特別的是標籤檔案需位於圖片檔案的上一層目錄，並且放置於 labels 資料夾底下，模型訓練前會自行對應圖片及標籤。

## ● 訓練過程

由於我們初始訓練檔案約有 28000 張圖片，圖片載入需要花費 3 小時，儘管將圖片對應成功後應該會產生 train.cache，但有時候又會抓不到快取檔，而且 colab 經常會異常斷線，因此在訓練上有些困難，我只成功訓練了 8 個 epochs，約耗時 5 小時。

## ● 減少訓練資料

由於初次訓練發現對於 cyclist 物件辨識度較差，且訓練過程耗時較久，因此我決定減少訓練資料，挑選出所有圖片中含有 cyclist 標籤的圖片，並將剩餘圖片隨機挑選，約只剩下十分之一的訓練資料，訓練時間便可以大幅減少，重新訓練 40 個 epochs 只需 2 個小時。



## ● 訓練結果比較



Fig. 37 訓練前後差異

Fig.37 左邊為初次訓練結果，可以發現 cyclist 物件標籤能力較差，Fig.37 右邊為修改訓練圖片後重新訓練，能夠看出明顯變化，不過對於腳踏車騎士及行人兩種物件，仍舊會發生誤判狀況。但我認為在這麼短的訓練時間，藉由載入 YOLOv5s 的預訓練權重，來訓練自己的圖片及標籤，YOLOv5 可以展現其強大的標籤能力及訓練速度。

## 5. Conclusion

## Conclusion

本次期末專題，我們這組選擇了 YOLO v3, v4, v5 來進行訓練以及比較，就 v3 和 v4 而言，v3 的訓練速度較 v4 來得快，但是 MAP, F1-score 等評比項目就比較差了一點，在同樣是 3000 iterations 的情況下，v3: MAP = 55.91%, F1-score = 0.65； v4: MAP = 70.87%, F1-score = 0.74。因此我們可以從結果來，v4 針對 v3 的 Darknet 53 架構上再加了一層 CSP Net，此舉也大大地讓 v4 表現更加突出了。

另外對於 v5 的訓練，我們在做 data loading 時花費的時間以及資源實在太多了，因此到最後所使用的訓練資料量只有其他演算法的十分之一，且訓練時間也不多，因此就不與其他兩者演算法做比較了，也許在未來能將 v5 的環境加以做改善，為自己建立一個友善的訓練環境，也才能與其他演算法做比較。



## 個人心得

陳彥銘:

由於我之前參加的比賽中有做過 yolo 的辨識，因此對於使用 yolo 來進行偵測辨識並不陌生，只須稍加回憶一下便能步入正軌，加上這次的資料是已經 label 過的，因此整體上不會像我之前必須花上大量時間找資料和進行標記，節省蠻多時間的。透過這次的期末專題，我覺得收穫更多的是在於了解到各 yolo 不同版本之間的差別演進以及相關策略，如果只是單純使用 yolo 來時做是不會特意去了解這些的，整體來說也是頗有收穫。

黃瑋翔:

在本次期末專題中，我負責的主要部份是做 Yolo v4 的研究以及訓練環境設置，在過程中研究了演算法背後的機制及基本的架構概念，另外剛開始在架設訓練環境時也遇到了些許困難，後來跟隊員討論之後也逐步地克服化解了問題，並成功將環境架設好並進行了訓練。不過由於我使用的訓練環境是 google colab，其免費版本對於 GPU 的用量有限制，因此我僅僅訓練到 1000 iteration 就停止了(中間也停過很多次)，後來就將剩餘的訓練部分交給擁有 colab pro 的大佬卓冠廷繼續進行訓練。那對於我來說，單看第 1000 次的訓練成果，v4 的表現是優於其他演算法的，從最後的結果也可驗證 v4 優於 v3，至於 v5 的訓練條件比較不同，比較起來就較不公平。

李惠軒:

在此次期末 project 我負責資料處理和 YoloV3 的部分處理資料時 tfrecord 轉換遇到困難，後改用解壓縮成 images 和 label.txt 的方式。

如果資料集不大使用 colab 還可以，但是 Waymo 資料集實在太龐大，Google Colab 的 VM 無法裝下太多包的 tar 檔，將資料放在 Google Drive 上透過 Mount 的方式來存取會出現效能瓶頸的問題，使用網路讀取資料的速度跟從硬碟讀取的速度根本不能比。由於 Colab 的 memory 有容量的限制因此在 Training 的時候 batch size 不能夠設太大，結果會不太好，

使用 Colab 做 YOLO，每次開啟 Colab 的 GPU 執行階段，分配到的 GPU 型號並不固定，而不同的 GPU 擁有的記憶體大小不同，會影響 batch、subdivisions 的設定，如果設置不當會出現 CUDA out of memory 的 error。改進的方式是會先檢查這一次開啟 GPU 執行階段分配到什麼樣的 GPU，如果剛好分到比較低階的(K80)，則試著重刷幾次看能不能換出比較高階的 GPU(T4 或 P100)來做使用。

透過調整 network input shape，希望能用較高解析度達到更好的辨識效果，但是卻忽略了 anchor box 的參數會隨之改變，而導致只能辨識出較小物件，若是測試圖像解析度低或是物件太大，則無法框選到。

然而在 Yolo v3 的訓練雖然相較於其他版本的 Yolo 訓練速度較快，但精準

度上卻還是有一定的差距，還有上述提到的 Yolo v3 的 anchor box 較其他版本較為不足，所以在圖像解析度低或是物件太大，則無法框選到。這一方面是可以著手去改善的。

卓冠廷:

在這 Final Project 實作過程中，我主要負責 Yolov4 部分，從訓練到圖片影像的辨識，架設環境和影片產生是使我最頭疼的部分，但也感謝組員幫忙做資料的前處理，我只需要去解決 Colab 環境的 Debug 即可，後續也有幫忙 Yolov3 的訓練實作，有了前面的考驗，v3 實作過程相對簡單。我也負責了 Yolov3、Yolov4 的實驗結果比較，不管從圖片或從影片或是效能評估的角度，都能發現兩者的一些落差，讓我能做更進一步的探討與研究，並從中找出問題去嘗試解決。

陳祈佑:

很感謝有這次物件偵測的實作 Project，儘管老師上課從頭講解了電腦視覺的演進與應用，不過我認為都比不上自己實作一次的經驗更難能可貴。

Yolo 這樣一代知名的物件偵測工具想必是多名偉大工程師的心血結晶和累積，就算我們無法完全理解其架構及演算法原理，但更重要是我們要學會如何善用這些優秀的工具。

我負責的是 Yolov5 的實作，在環境建置過程中也遇到不少問題，不過在逐一解決 bug 的過程中我對 python 語法以及自行查詢資料的能力也獲得了提升，而最後的結果雖然不盡完美，卻讓我體會到了 Yolo 的強大之處，我未來也會對相關領域的進步和創新持續追蹤並學習。

張智彥:

原本嘗試以 Fast R-CNN 的 model 做 Waymo 的 2D object detection，在網路上搜尋可以使用的 code 時，在把需要的版本的套件都安裝完後，執行程式碼時，還是有 error。邊 google 邊安裝缺少的套件，發現有一些套件的版本之間不相容，error 解不了之後這部分不了了之，最後只剩下 YOLO 的 model。希望以後有機會嘗試更多不同的模型及訓練方法。

## 6. Reference

- [ 1 ] [Waymo open Dataset](#)
- [ 2 ] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in CVPR, 2014.
- [ 3 ] R. Girshick, “Fast r-cnn,” in ICCV, 2015.
- [ 4 ] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in NIPS, 2015, pp. 91–99.
- [ 5 ] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in CVPR, 2016.
- [ 6 ] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in ECCV, 2016.
- [ 7 ] YOLO 簡介  
<https://mropengate.blogspot.com/2018/06/YOLO-YOLO v3.html>
- [ 8 ] Real-time object detection and classification  
<https://github.com/thtrieu/darkflow>
- [ 9 ] Object detection 的測試實作  
<https://towardsdatascience.com/YOLO-object-detection-with-opencv-and-python-21e50ac599e9>
- [ 10 ] Object detection for darknet  
<https://github.com/Garima13a/YOLO-Object-Detection/blob/master/YOLO.ipynb>
- [ 11 ] Darknet 官網  
<https://pjreddie.com/darknet/>
- [ 12 ] YOLOv3——引入：FPN+多尺度检测（目标检测）(one-stage)(深度学习)(CVPR 2018)  
[https://paperswithcode.com/method/cspdensenethttps://blog.csdn.net/Gentleman\\_Qin/article/details/84350496](https://paperswithcode.com/method/cspdensenethttps://blog.csdn.net/Gentleman_Qin/article/details/84350496)
- [ 13 ] <https://paperswithcode.com/method/cspdensenet>
- [ 14 ] <https://www.nature.com/articles/s41598-021-81216-5>
- [ 15 ] <https://medium.com/clique-org/panet-path-aggregation-network-in-yolov4-b1a6dd09d158>
- [ 16 ] <https://github.com/ultralytics/yolov5/issues/5541>
- [ 17 ] Yolov5 資源 <https://github.com/ultralytics/yolov5>