

國立成功大學 測量及空間資訊學系

專題研究（一）成果報告

題目：Yolo 影像辨識演算法之演進與實作

學生：吳冠志

指導老師：林昭宏 教授

# 目錄

摘要 .....	1
Yolo 簡介 .....	2
Yolo 類神經網路 .....	2
Yolo 評估指標 .....	2
Yolo v1 .....	3
物件偵測原理 .....	3
網路架構 .....	3
Loss function .....	4
Non-Maximum Suppression(NMS) .....	4
Yolo v1 缺點 .....	5
Yolo v2 .....	5
Yolo v2 較 v1 改進 .....	5
Batch Normalization .....	6
High Resolution Classifier .....	6
Anchor Box .....	6
Dimension clusters .....	7
Darknet-19 .....	7
Direct location prediction .....	8
Fine-Grained Features .....	8
Multi_Scale Trainging .....	9
Yolov3 .....	10
Bounding Box prediction .....	10
Prediction across scals .....	10
Feature Extractor .....	10
Yolov4 .....	10
BACKBONE: CSPDARKNET53 .....	10
Neck: SPP+PAN .....	10
HEAD: YOLO HEAD .....	11
Mosaic data augmentation .....	12
DropBlock regularization .....	12
Mish activation .....	13
IOU loss .....	13
Self-Adversarial Training(SAT) .....	14

成果 1.....	15
Dataset 準備.....	15
LabelImg.....	15
準備資料.....	15
Colab 程式碼.....	15
成果.....	17
成果 2.....	18
訓練資料.....	18
訓練資訊.....	18
成果.....	19
影片成果.....	19
結論.....	20
參考文獻.....	21

## 摘要

深度學習是近年來備受關注的技術，在許多領域都陸續投入使用，學術研究上技術也是日星月異，許多各式各樣的模型如雨後春筍般冒出，並在許多領域都有優異的表現。其中一項技術是物件辨識及分類，此技術可以追蹤到移動中的物體，並透過已知的資料及加以分類，我們對此技術深感興趣並在本次專題嘗試深入了解。

YOLO ( You Only Look Once: Unified, Real-Time Object Detection )，是基於單個神經網路 ( one stage ) 的目標檢測系統，於 2015 年提出，近幾年在物件偵測及分類有相當優異的表現。本次專題嘗試了解 yolo 之原理、分析不同 yolo 版本及模型之差異並嘗試架構訓練自己的模型。

# 介紹

## ● Yolo 簡介

### Yolo 類神經網路

YOLO (You Only Look Once) 是一個 one-stage 的 object detection 演算法，將整個影像輸入只需要一個 CNN 就可以一次性的預測多個目標物位置及類別，這種 end-to-end 的算法可以提升辨識速度，能夠實現 real-time 偵測並維持高準確度。

### Yolo 的評估指標

YOLO 的評估指標主要採取 IOU 和 mAP

IOU:指的是 predict 的 boundingbox 與 Ground Truth 的交集除以聯集，可參考下圖

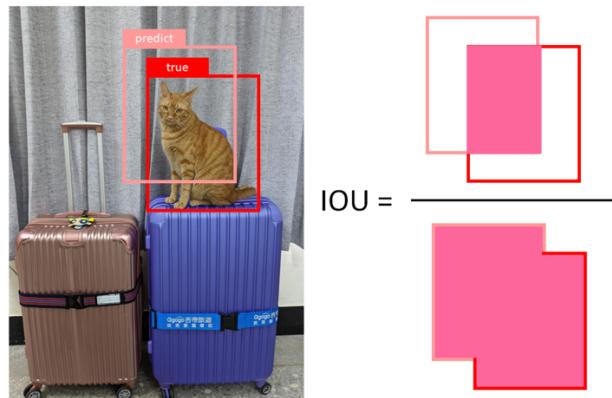


圖 [1] IOU 說明

mAP:各類 AP 的平均值，其中 AP 指的是 PR curve 的面積，PR curve 是以 recall 為 x 軸,Precision 為 y 軸所繪製之曲線，兩者越高代表效能越好

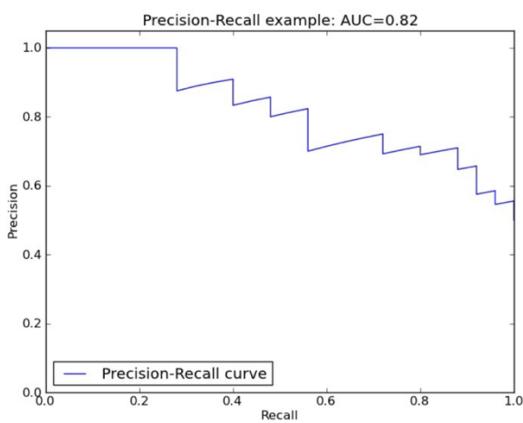


圖 [2] mAP 示意圖

其中 Precision ( 準確率 ) =  $TP / (TP + FP)$

Recall ( 召回率 ):  $TP / (TP + FN)$

TP (True Positive): 實際為目標物件，也正確地預測出是目標物件

TN (True Negative): 實際不為目標物件，也正確地預測出不是目標物件

FP (False Positive): 實際不為目標物件，但卻錯誤地預測成是目標物件，也稱作 Type 1 Error

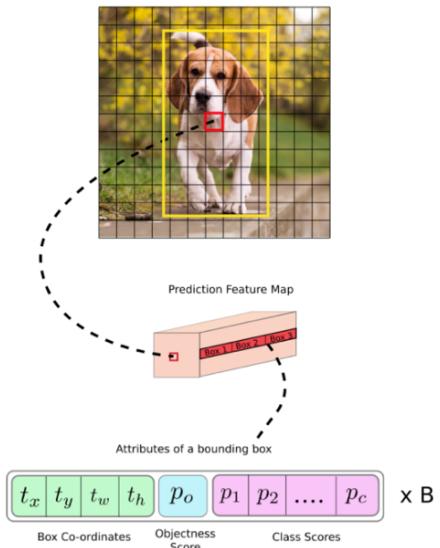
FN (False Negative): 實際為目標物件，但卻錯誤地預測成不是目標物件（或是指沒預測出來的正樣本），也稱作 Type 2 Error

## ● Yolo v1

### 物件偵測原理

Yolo v1 想法是將整張照片作為輸入，直接預測物體的位置及分類。首先將輸入影像切割成  $S \times S$  的網格 (grid)，若被偵測的物體中心落於某網格內，該網格就須負責偵測該物體。而每個網格負責預測  $B$  個 bounding box 和  $C$  個類別（目標屬於哪種物體），而每個目標又分別輸出 5 個預測值：外框中心點  $x,y$ 、物體快框大小  $w,h$  以及信心值 confidence，因此最終輸出  $S \times S \times (B+5+C)$ ，如左圖 [1]

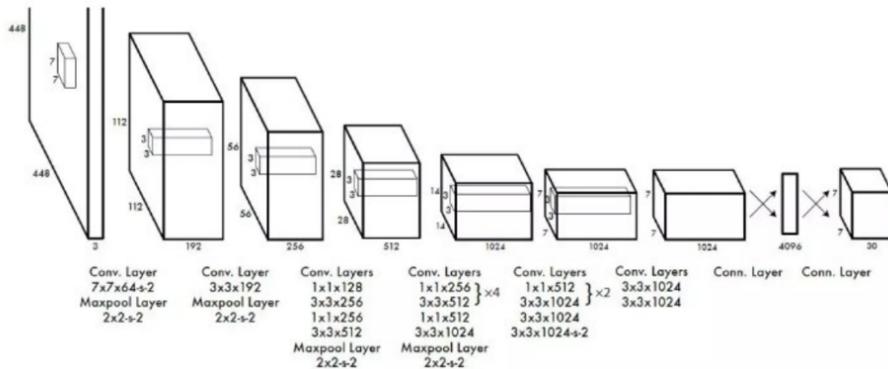
Image Grid. The Red Grid is responsible for detecting the dog



圖[3] 輸入為一張圖片 而輸出為  $S \times S \times (B+5+C)$  的網路架構

### Yolo v1 網路架構

將輸入統一縮放為 448\*448 大小，對圖片做卷積神經網路 (Convolutional Neural Network,CNN)，其網路架構參考 Google Net，經過 24 個卷積層及 2 個全連接層後，輸出  $7 \times 7 \times 30$  的區域（假設將圖片切割成  $7 \times 7$  隻區域，並預測 20 個類別），網路架構如下圖



圖[4] yolo v1 的網路架構

### Loss Function:

一般使用的平方誤差和不能完美校正最大化目標的平均精度，因為其中每項 error 占有一樣的比重，修正後的 loss function 如下圖

$$\begin{aligned}
 & \text{判斷第} i \text{個網格中第} j \text{個bbox} \\
 & \text{是否負責這個object} \quad \text{座標預測} \\
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad \text{對包含object的bbox} \\
 & \text{做confidence預測} \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad \text{對不含object的bbox} \\
 & \text{做confidence預測} \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad \text{類別預測} \\
 & \text{判斷是否有Object中} \\
 & \text{心落在第} i \text{個網格中}
 \end{aligned}$$

圖 [5] 上圖為 yolov1 loss function 解釋圖，分為座標預測，confidence 預測及類別預測

其中 包含物件的 bbox 傳播損失權重  $\lambda_{\text{coord}}=5$ 、不包含物件的 bbox 傳播損失權重  $\lambda_{\text{noobj}}=0.5$ 、並以對 w、h 取平方根的操作降低因 bbox 大小而據不同影響力的 bias

### Non-Maximum Suppression(NMS):

在物件偵測過程中，一個物件可能被很多候選框選到，yolo 則使用 Non-Maximum Suppression 的方法消除多餘物件框找到最佳的框。

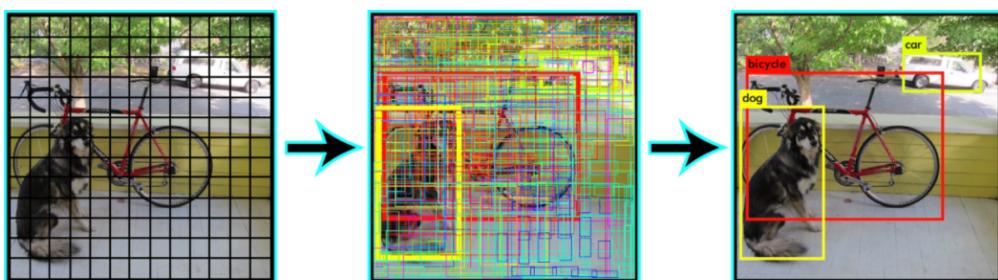


圖 [6] NMS 示意圖，嘗試從多個候選框找出最符合真值的框

選出來的框叫 bounding box ( BBox )，每個框有五個值，分別為框中心(x,y)長寬(w,h)及一個 confidence score，下圖為 NMS 演算法

```

Input :  $\mathcal{B} = \{b_1, \dots, b_N\}$ ,  $\mathcal{S} = \{s_1, \dots, s_N\}$ ,  $N_t$ 
 $\mathcal{B}$  is the list of initial detection boxes
 $\mathcal{S}$  contains corresponding detection scores
 $N_t$  is the NMS threshold

begin
     $\mathcal{D} \leftarrow \{\}$ 
    while  $\mathcal{B} \neq \text{empty}$  do
         $m \leftarrow \operatorname{argmax} \mathcal{S}$ 
         $\mathcal{M} \leftarrow b_m$ 
         $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$ 
        for  $b_i$  in  $\mathcal{B}$  do
            if  $iou(\mathcal{M}, b_i) \geq N_t$  then
                 $\mid \mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$ 
            end
        end
    end
    return  $\mathcal{D}, \mathcal{S}$ 
end

```

圖 [ 7 ] Non-Maximum Suppression 演算法

其步驟大致如下：

- 先看哪個 BBOX 的信心程度最高，那個 BBOX 會進去「確定是物件集合」內
- 其他 BBOX 和剛選出來的 BBOX 算 IoU，然後算出來的 IoU 大於設定好的閾值的 BBOX，那些 BBOX 的信心度會被設定為 0

重複上述兩步驟直至所有 bounding box 都被處理完，則可得最後「確定是物件集合」的最後結果

### Yolo v1 缺點

Yolo v1 帶來每秒 45 張影格的運算速度，可達到實時運算的速度，但仍有其局限性：

一個格子只預測兩個框，一個框子給出一個分類，若同一格子出現多個小物體 ( ex: 一群人或一群鳥 ) 會偵測不出  
框很粗糙 ( 長方形 )，對於不規則物體泛化能力較差差

小的錯誤在小的格子中影響很大，因為在 loss function 中，定位誤差是影響檢測效果的主因

### ● Yolo v2

#### Yolov2 較 v1 的改進

Yolo v1 雖然檢測速度快，但精度卻不如 R-CNN 係檢測方法，yolo v2 共提出幾種方法提升 Precision 及 Recall，從而提升 mAP，從下圖 [ 8 ] 可看出 yolo v2 mAP 從原本 63.4% 提升到 78.6%

	YOLO	YOLOv2
batch norm?	✓	✓
hi-res classifier?	✓	✓
convolutional?	✓	✓
anchor boxes?	✓	✓
new network?	✓	✓
dimension priors?	✓	✓
location prediction?	✓	✓
passthrough?	✓	✓
multi-scale?		✓
hi-res detector?		✓
VOC2007 mAP	63.4	65.8 69.5 69.2 69.6 74.4 75.4 76.8 <b>78.6</b>

圖 [ 8 ] yolo v2 改進策略

## Batch Normalization

可提升模型收斂速度，並起到正則化效果，降低模型過擬和。Yolo v2 在每個卷積層後都添加 Batch Normalization，並不再使用 dropout

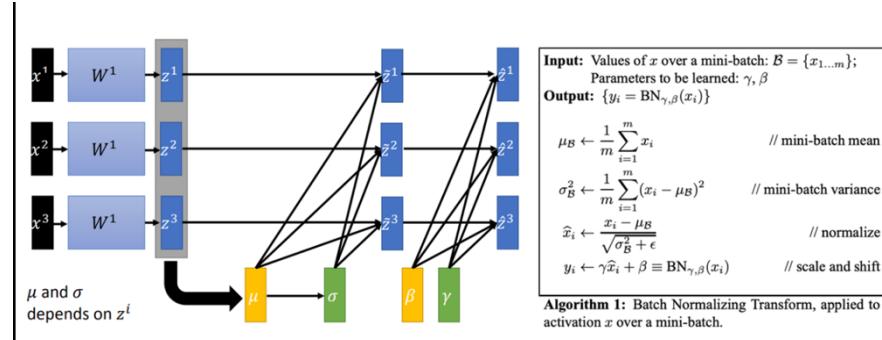


圖 [9] batch Normalization 演算法示意圖，對每層網路歸一化收斂更快

## High Resolution Classifier

Yolo v1 使用之圖片解析度為 224\*224 不利於檢測模型，因此 yolo v2 將分類器級 ImageNet 資料集解析度增加至 448\*448

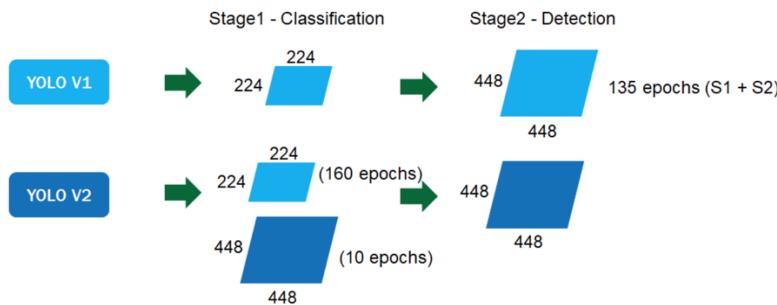


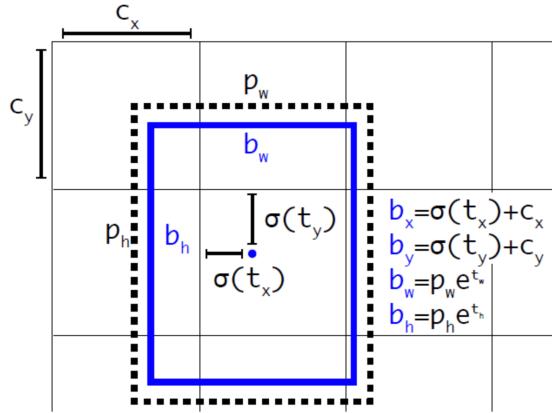
圖 [10] High Resolution Classifier 示意圖，將預訓練分為兩步

如上圖所示，v1 在 stage 1 時採用 224\*224 resolution 做訓練分類網路，在 Stage2 的時候以 448\*448 的 resolution 訓練檢測網路，造成 YOLOv1 的捲積層要重新適應新的分辨率同時 YOLOv1 的網路還要學習檢測網路

因此 yolo v2 提出一個方法，在 stage 1 時先用低解析度訓練 160 個 epoch，在用 10 個 epoch 訓練網路適應 448\*448 高解析度，最後讓 stage2 以相同解析度訓練

### Anchor Box(先驗框策略):

由 Fast RCNN 導入，不同分類的物體在圖片中的比例及大小皆不同（行人邊界框偏高瘦、而車輛則可能呈現矮胖外型），Anchor box 一般以手工指定 5 到 10 個形狀，並使用 K-Means 求 anchor box 比例，不再直接 mapping bounding box 的座標，而是預測每個 anchor 的 bbox 以及類別。在使用 Anchor box 後 YOLOv2 可以預測上千個邊界框 ( $13*13*\text{num\_anchors}$ )，相較 YOLO v1 的  $7*7*2$  98 個邊界框有大幅提升，使用 Anchor Box 後 mAP 值些微下降，但 Recall 與 Yolo v1 相比，從 81% 提升到 88%，有相當顯著的提升。



圖[11] Anchor box 示意圖，預測每個 anchor 的 bbox 及類別，預測量大幅提升

### Dimension clusters

在 Fast R-CNN 中 anchor box 大小和比例是靠經驗設定的，yolo v2 作者採用 k-means 方式對訓練集 bounding boxes 作聚類，試圖找到更合適的 anchor boxes，以幫助網路預測 detection

### Darknet-19:

Yolo v2 使用新的基礎模型，稱 Darknet-19，包含 19 個卷積層和 5 個 maxpooling 層，不再使用全連接層，主要採用  $3 \times 3$  卷積， $2 \times 2$  maxpooling 層，在  $3 \times 3$  卷基層之間使用  $1 \times 1$  卷積層來壓縮以降低模型計算量，且每個模型後同樣添加 Batch Normalization，降低模型過擬和。

使用 Darknet-19 後 mAP 值無顯著提升，但計算量減少約 33%

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

圖[12] Darknet-19 網路架構

### Direct location prediction

Yolo v2 提出新的預測公式，對 anchor box 點的偏移位置進行預測，每個 bbox 預測五個值： $t_x$ 、 $t_y$ 、 $t_w$ 、 $t_h$ 和 $t_o$ 如下

$c_x$ ， $c_y$  為框的中心座標所在的 grid cell 距離左上角第一個 grid cell 的 cell 個數。

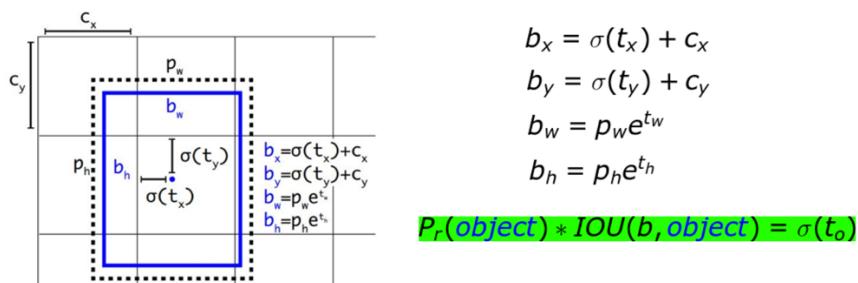
$t_x, t_y$  為預測的邊框的中心點座標。

$\sigma()$ 函數為 logistic 函數，將座標歸一化到 0-1 之間。最終得到的  $b_x, b_y$  為歸一化後的相對於 grid cell 的值

$t_w, t_h$  為預測的邊框的寬，高。

$p_w, p_h$  為 anchor 的寬，高。實際在使用中，作者為了將  $b_w, b_h$  也歸一化到 0-1，實際程序中的  $p_w, p_h$  為 anchor 的寬，高和 featuremap 的寬，高的比值。最終得到的  $b_w, b_h$  為歸一化後相對於 anchor 的值

$\sigma(t_o)$ 表示預測的邊框的置信度，為預測的邊框的概率和預測的邊框與 ground truth 的 IOU 值的乘積。



圖[13] anchor box with Direct location prediction

Sigmoid 函數約束在(0,1)範圍內，因此預測邊框中心點被約束在藍色網格內，約束邊框使得模型更容易學習，且預測更穩定

公式以 e 為底以 scale 放大率為次方，計算出數值的自然放大率

### Fine-Grained Features

卷積層逐漸減小空間維度，隨著分辨率降低，檢測小物體變得困難，因此為了抓取細粒度特徵，yolo v2 加入 PassThrough layer(又稱 reorg layer)將原本  $26*26$  的 resolution 的 feature map 進行特徵重排，再與原始  $13*13*1024$  輸出層連接得  $13*13*3072$  的 layer 再使用 convolution filters 進行預測，如下圖

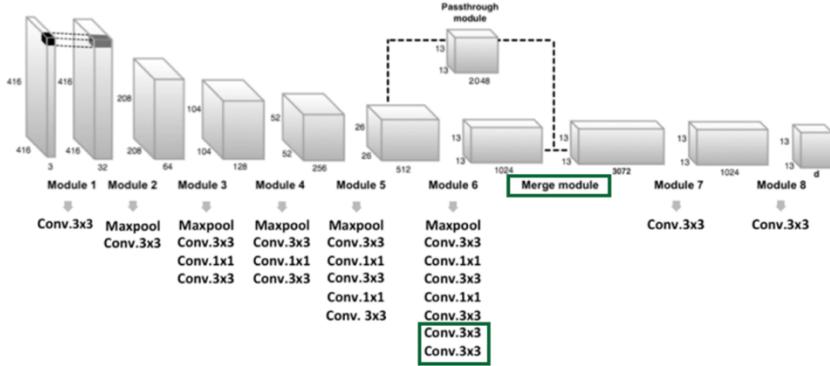


圖 [14] Fine-Grained Features 結構，透過 passThrough layer 增加 layer 的 channel 維度

### Multi\_Scale Trainging

為了讓模型更穩定，Yolo V2 做了多尺度訓練，每 10 個 epochs 隨機選擇一種輸入圖片大小，然後只需修改對最後檢測層的處理就可以重新訓練

由於 YOLOv2 的轉換層將輸入維度下採樣 32 倍，因此新採樣的大小為 32 的倍數，參考下圖

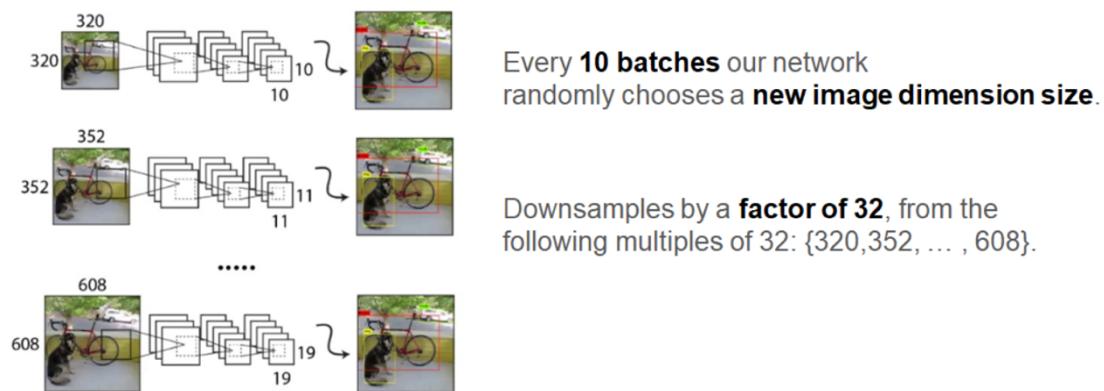


圖 [15] Multi-Scale Training，透過此法可以適應不同大小圖片，並預測出很好的結果

### 1. For Detection

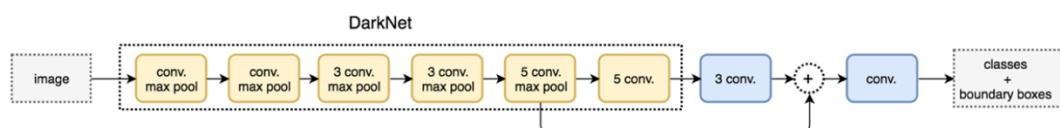


圖 [17] Yolo v2 網路架構 (for detection)

去掉 Darknet-19 最後的卷積層並加入 PassThrough Layer

Fine tune 這個預訓練模型 160 個 epoch

調整學習率，在第 60 次和第 90 次的 epoch 學習率減為原來的 1/10

$$lr = \begin{cases} 10^{-3}, & epoch = 1 \\ 10^{-4}, & epoch = 60 \\ 10^{-5}, & epoch = 90 \end{cases}$$

Weight Decay 為 0.0005

Momentum 為 0.9

### ● Yolov3

Yolo v3 相較於 yolo v2 改動並不多，但仍然帶來了不錯的性能提升

改動包括邊界預測分數、多標籤分類、預測多尺寸邊界框、使用不同網路架構

Bounding Box prediction:

Yolo v3 繼續採用前面版本利用 anchor box 預測 bounding box 的方法，但其還為每個邊界框根據預測框與物體的重疊度預測了一個分數 objectness score，若某框重疊度較其他框高，其分數為一，忽略那些不是最好且重疊度大於某一閥值的框，可以減少計算量

Prediction across scales:

Yolo v3 預測三個尺寸檢測框。因此最後輸出三維邊界框（四個偏移量），分數級 80 個類別(一使用 dataset 而定)，因此輸出向量為  $N*N*[3*(4+1+80)]$

Feature Extractor:

特徵提取使用了新網路，因其有 53 個卷積層，稱其為 Darknet-53

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	<b>171</b>
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	<b>77.6</b>	<b>93.8</b>	29.4	1090	37
Darknet-53	77.2	<b>93.8</b>	18.7	<b>1457</b>	78

Table 2. Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

圖 [18] 比較各種網路精度、FPS 等數值，可看出 Darknet 擁有與 ResNet 相似的性能，並且快兩倍由於加深網路層數，Darknet-53 比 Darknet-19 慢的許多，但 Darknet-53 處理速度每秒 78 張圖，還是比同精度的 ResNet 快很多，YOLOv3 依然保持了高性能

### ● Yolov4

Yolo v4 對 v3 的各個部分做改進，在能保證速度的同時，大幅提升檢測精度

BACKBONE: CSPDARKNET53

目標檢測任務中常是運行在小型設備上，需要計算量較低之模型，以縮小預測時間

CSPNET 主要目的是使網路架構能實現獲取更豐富梯度融合信息並降低計算量，先將 FEATURE MAP 分成兩部分，經過不同 LAYER 在進行融合 (CROSS STAGE PARTIAL DENSENET)

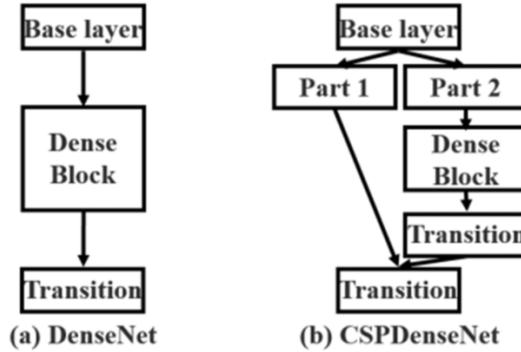


圖 [ 19 ] DenseNet,CSPDenseNet 網路結構比較

可使在 imageNet 分類準確率不便或略為提升，計算量卻大幅下降

**Neck: SPP+PAN**

SPP(Spatial Pyramid Pooling):在最後一層 concate 所有 feature map，後面能繼續接 CNN module

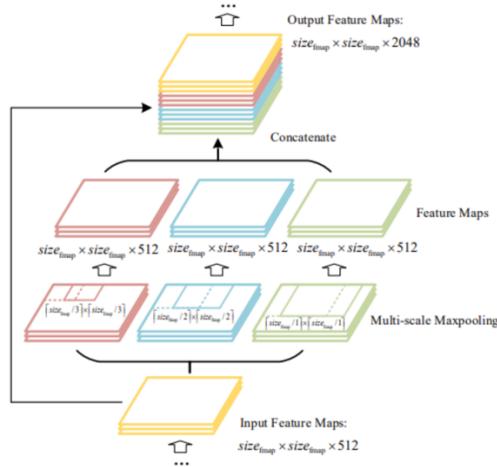
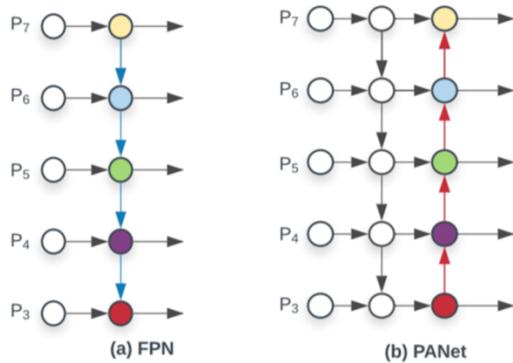


圖 [ 20 ] SPP 網路結構

PANet(Path Aggregation Network):以 FPN 為基礎做改進，將串的層數再多加一層



圖[20] PANet 網路結構

HEAD: YOLO HEAD : 沿用 yolov3

## Mosaic data augmentation

採用隨機縮放、裁減方式混合拼接 4 種圖片進行訓練，是有效進行數據增強的方式

採用此種方式可以豐富檢測數據集，也因增加許多小目標，讓模型穩健性更好

## DropBlock regularization

避免 overfitting 之一種正則化方法，與 Dropout 類似

使用 Dropout 方式隨機刪除神經元數量，網路仍可從相鄰激活單元學習到相同訊息

Dropblock 隨機將整個局部區域進行刪減，網路會注重學習某些特徵以實現正確分類達到更好的泛化效果

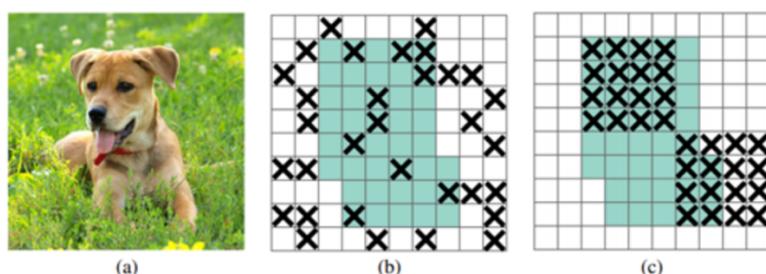


Figure 1: (a) input image to a convolutional neural network. The green regions in (b) and (c) include the activation units which contain semantic information in the input image. Dropping out activations at random is not effective in removing semantic information because nearby activations contain closely related information. Instead, dropping continuous regions can remove certain semantic information (e.g., head or feet) and consequently enforcing remaining units to learn features for classifying input image.

圖「21」DropBlock 原理，將局部區域刪減達到更好效果

### Mish activation

和 ReLU 相比，Mish 的梯度更平滑，且在負值時允許有較小的負梯度，可以穩定網路梯度流，具有更好的泛化能力

$$f(x) = x \tanh(\ln(1 + e^x))$$

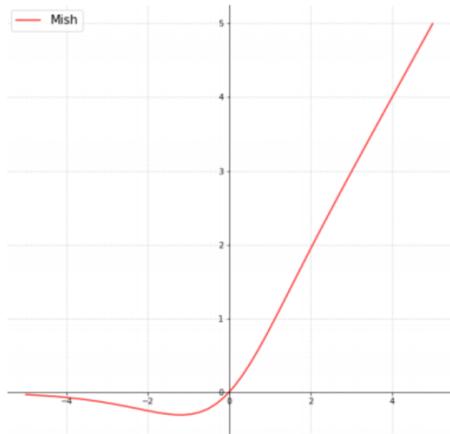


Figure 1. Mish Activation Function

圖 [22] Mish 激活函數

Yolo v4 在 Backbone 中使用 Mish 激活函數，後面網路則仍使用 leaky relu 函數

### IOU loss

#### 缺點

當預測匡和目標匡不相交時，IOU 為 0 無法反映兩匡之間的距離，因此無法優化不相交之情形

IOU 無法反映預測匡與目標匡重合大小

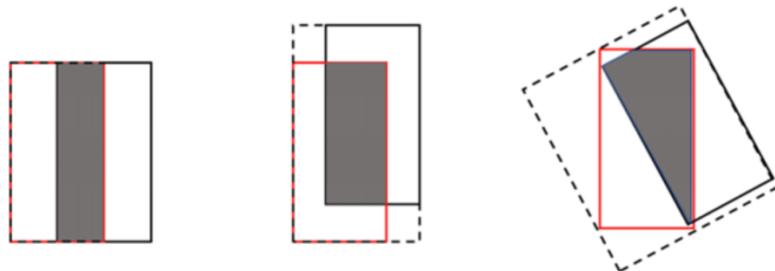


圖 [23] 三種圖擁有相同 IOU 值但重合度不同

CIOU(complete IOU)

$$L_{CIOU} = 1 - IOU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v,$$

$$v = \frac{4}{\pi^2} (\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h})^2,$$

$$\alpha = \frac{v}{(1 - IOU) + v}$$

$b, b^{gt}$  分別表示  $B, B^{gt}$  的中心點,

$\rho$  是歐式距離,

$c$  是最小包圍兩個框的對角線長度,

$\alpha$  是權重函數, 依據兩個框之間的距離來調整,

$v$  是衡量長寬比的相似性

圖 [24] CIOU 之計算公式

考慮與 ground truth 之重疊面積、中心點距離及長寬比

Self-Adversarial Training (SAT)

先對訓練樣本進行前向傳播，然後進行反向傳播時修改圖片像素，將低模型檢測性能，增加樣本訓練難度

接著對修改過的圖片對模型做訓練

有助於推廣模型和降低 overfitting 問題

## ● 成果展示 1

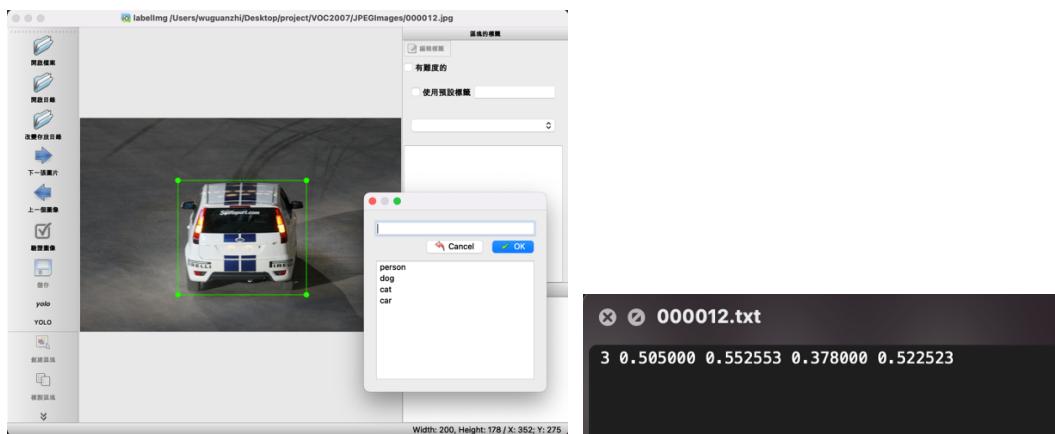
使用官網提供之網路架構 darknet53.conv.74，嘗試透過 colab，使用自己的資料集訓練權重並輸入圖片測試分析成效

## 1. Dataset 準備

本次訓練資料來源來自 VOC 2007，並只對 person、dog、cat、car 四種圖片庫出現較頻繁的資料做處理及預測

## 2. LabelImg

從 <https://github.com/tzutalin/labelImg> 下載之軟體，可對圖片中物件標示並分類，以便機器學習



透過上述方式可對圖片物件做標記，並輸出成 txt 檔，本次資料共有 182 張圖片，以及其對應標記的 txt 檔，因本次要利用 colab 做訓練，因此將所需要的資料存入雲端中

### 3. 準備資料：

`labelled_data.data`: 類似檔案目錄, 用來告訴 darknet 這套深度學習框架, 其他用來訓練的檔案資源在哪、偵測物件的類別數量以及訓練權重該備份到哪裡去等

class.names:告訴 darknet 分類各類別名稱，本次分為 people、dog、cat、car 四種 class

Yolov3-tiny.cfg: 告訴 darknet 模型架構長得怎樣，需小改部分參數

train.txt 及 test.txt 則是用來告訴 darknet 訓練及驗證用檔案存在哪裡

## 4. Colab 程式碼

```
[1] from google.colab import drive  
drive.mount('/content/gdrive')  
  
Mounted at /content/gdrive
```

將 colab 與雲端連線

```
git clone 'https://github.com/AlexeyAB/darknet.git' '/content/gdrive/My Drive/space_colab/darknet'
Cloning into 'content/gdrive/My Drive/space_colab/darknet...'...
remote: Counting objects: 1004, done.
remote: Compressing objects: 1004 (37/37), done.
remote: Total 15132 (delta 26), reused 44 (delta 20), pack-reused 15073
Receiving objects: 100% (15132/15132), 13.50 MiB | 7.84 MiB/s, done.
Resolving deltas: 100% (10272/10272), done.
Checking out files: 100% (2036/2036), done.
```

從 github 上下載 darknet  
至 google drive

```
cd '/content/gdrive/My Drive/space_colab/darknet'
!make

gcc -Iinclude/-I3rdparty/stb/include -Wall -Wfatal-errors -Wno-unused-
./src/yolo_layer.c: In function 'process_batch':
./src/yolo_layer.c:426:25: warning: variable 'best_match_t' set but not
int best_match_t = 0;
```

將 Makefile 中的 GPU、  
CUDNN 及 OPENCV 設成 1  
並編譯

[7] !python Yolos/creating-files-data-and-name.py

產生 labelled\_data.data

[ ] !python Yolos/creating-train-and-test-txt-files.py

以 0.85:0.15 的比例切分  
training data 以及  
testing data

[ ] !chmod +x ./darknet/darknet

複製之前編譯過的 darknet

此時我們有所有訓練時需要的檔案了，包含 train.txt、test.txt、  
labelled\_data.data、yolov3.cfg 及 darknet53.conv.74 我們就可以開始訓練了

```
./darknet/darknet detector train ./Yolos/labelled_data.data ./Yolos/yolov3-tiny.cfg ./Yolos/darknet53.conv.74 -dont_show
total_boxs = 1550, rewritten_boxs = 0.440000 %

32: 424.876556, 365.660126 avg loss, 0.000000 rate, 0.314654 seconds, 768 images, 67.160848 hours left
Loaded: 0.000024 seconds
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.535880), count: 6, class_loss = 170.570694, iou_loss = 2.022349, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.540652), count: 4, class_loss = 676.016418, iou_loss = 1.078695, total_lo
total_boxs = 1608, rewritten_boxs = 0.435323 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.547018), count: 5, class_loss = 170.237686, iou_loss = 0.951243, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.831046), count: 1, class_loss = 675.016235, iou_loss = 0.071208, total_lo
total_boxs = 1614, rewritten_boxs = 0.433705 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.686594), count: 5, class_loss = 169.821365, iou_loss = 0.822906, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.682129, iou_loss = 0.000000, total_lo
total_boxs = 1619, rewritten_boxs = 0.432366 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.496829), count: 6, class_loss = 170.573105, iou_loss = 2.356293, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.684387, iou_loss = 0.000000, total_lo
total_boxs = 1625, rewritten_boxs = 0.430769 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.552578), count: 5, class_loss = 170.155701, iou_loss = 1.430400, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.685364, iou_loss = 0.000000, total_lo
total_boxs = 1630, rewritten_boxs = 0.429448 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.575988), count: 4, class_loss = 169.485794, iou_loss = 0.755503, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.686462, iou_loss = 0.000000, total_lo
total_boxs = 1634, rewritten_boxs = 0.428397 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.525075), count: 8, class_loss = 171.241272, iou_loss = 3.149719, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.686707, iou_loss = 0.000000, total_lo
total_boxs = 1642, rewritten_boxs = 0.426309 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.520379), count: 5, class_loss = 170.240158, iou_loss = 1.972697, total_lo
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 674.685974, iou_loss = 0.000000, total_lo
total_boxs = 1647, rewritten_boxs = 0.425015 %
```

對我們的資料及做訓練，以 darknet53 作為深度學習框架，並將結果權重存至  
backup 資料夾中

```
!./darknet/darknet detector train ./Yolos/labelled_data.data ./Yolos/yolov3-tiny.cfg ./backup/yolov3-tiny_last.weights -dont_show
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 0.000040, iou_loss = 0.000000, total_loss
total_bbox = 9756, rewritten_bbox = 0.440754 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.768260), count: 5, class_loss = 0.386169, iou_loss = 0.248022, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.855746), count: 1, class_loss = 0.017916, iou_loss = 0.038834, total_loss
total_bbox = 9762, rewritten_bbox = 0.440484 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.776506), count: 4, class_loss = 0.572568, iou_loss = 0.199343, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.693685), count: 12, class_loss = 1.709485, iou_loss = 1.222341, total_loss
total_bbox = 9778, rewritten_bbox = 0.439763 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.771104), count: 6, class_loss = 1.080674, iou_loss = 0.330498, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 0.000000, iou_loss = 0.000000, total_loss
total_bbox = 9784, rewritten_bbox = 0.439493 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.721336), count: 5, class_loss = 0.412381, iou_loss = 0.451274, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.690797), count: 2, class_loss = 0.593786, iou_loss = 0.121388, total_loss
total_bbox = 9791, rewritten_bbox = 0.439179 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.850406), count: 5, class_loss = 0.081606, iou_loss = 0.144512, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.000000), count: 1, class_loss = 0.000640, iou_loss = 0.000000, total_loss
total_bbox = 9796, rewritten_bbox = 0.438955 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 16 Avg (IOU: 0.660148), count: 2, class_loss = 0.052011, iou_loss = 0.217635, total_loss
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 23 Avg (IOU: 0.500280), count: 19, class_loss = 2.717210, iou_loss = 2.539397, total_loss
...
```

若下次想繼續訓練，可將前一次訓練之 yolov3-tiny-last.weights 加入指令，

可從上次訓練進行的地方繼續訓練，每 10000 個

接著將下面照片（檔名：2007\_000423.jpg），置入模型中做物件偵測



```
!./darknet/darknet detector test ./Yolos/labelled_data.data ./Yolos/yolov3.cfg ./backup/yolov3-tiny_last.weights ./2007_000423.jpg -dont_show
*** YOLO ***
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
Total BFLOPs: 139.543
avg_outputs = 1105044
Allocate additional workspace_size = 52.43 MB
Loading weights from ./backup/yolov3-tiny.last.weights...
seen 64, trained: 288 K-images (4 Kilo-batches_64)
Done! Loaded 48 layers from weights-file
Detection layer: 82 - type = 28
Detection layer: 94 - type = 28
Detection layer: 106 - type = 28
./2007_000423.jpg: Predicted in 70.499000 milli-seconds.
```

透過前個步驟產生之權重檔對照片做物件偵測

接著我們透過函式，將偵測後之結果呈現在 colab 介面上



## 5. 成果

完全沒有偵測到任何物件，可能原因為訓練樣本太少（本次使用 182 張相片當訓練資料）及訓練時間太短，或許未來增加新的資料及和花更久時間訓練後會有不錯的成果。

### ● 成果展示 2- 使用 yolo v3 即時口罩物件偵測

#### 1. 訓練資料

來自kaggle的Data，共有mask及non-mask兩類

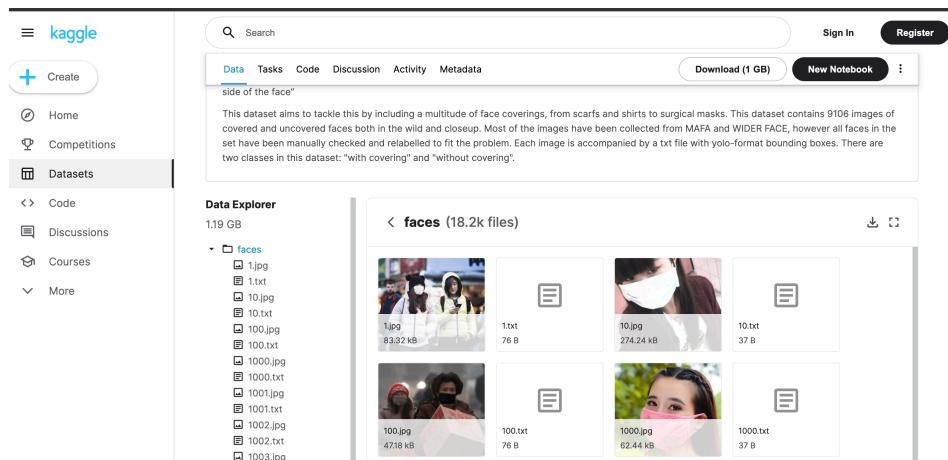


圖 [ 25 ] kaggle口罩訓練資料( 資料來源網址：<https://www.kaggle.com/karm1a/covid19-face-coverings-at-yolov4-format?fbclid=IwAR1B424XUUEvR1to91WkYg43v7CW6dxxVr1M-qrmSNqir1shUj2wq0pAPs> )

本次測試資料數目共1321筆，訓練資料共7486筆

有口罩樣本共6776筆，無口罩樣本共10291筆

#### 2. 訓練資訊

Batch: 第 6052 組

總損失：1.042419

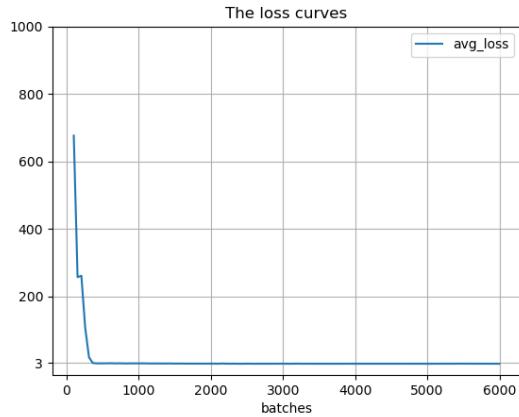
平均損失：1.022733

學習率：0.01000

當前 batch 訓練所花時間：0.747029seconds

目前參與訓練圖片數：145248

### 3. 成果



圖[26] batch 及與之對應的 average IOU loss

由上圖可看出，總共訓練約 6000batch，但 loss 在不到 1000batch 就急速下降，代表此模型很快就收斂，達到很好偵測效果



圖[27] 透過本模型偵測含口罩物件之圖片

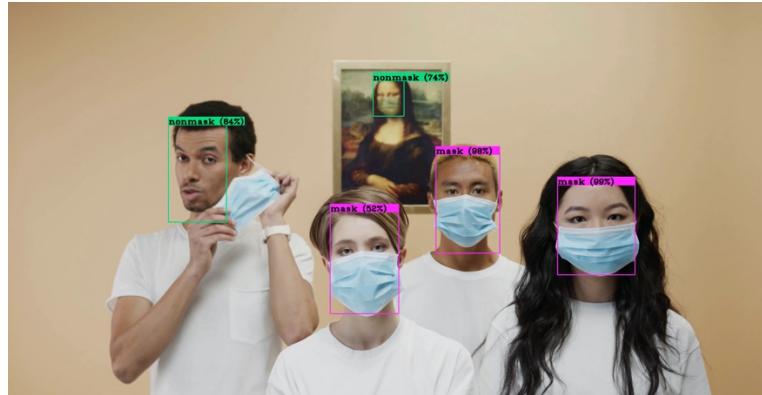
從上圖可看出，大部分戴口罩行人都被偵測出，但仍有一名男性其未被偵測出戴口罩，代表若需被偵測之物件數量多或被偵測物件重疊時，此模型可能會有缺漏

### 4. 影片成果

由於影片成果難用報告展示，因此以多張圖片進行成果呈現



圖[28] 偵測函口罩物件之影片



圖[29] 偵測函口罩物件之影片

上兩張圖為本次 model 偵測影片出來的結果，可以看出結果大致符合預期，不論是帶著口罩還是中途把口罩拿下來模型都可將其正確分類為 mask 及 nonmask，但模型卻將後面的蒙娜麗莎畫像也進行了分類，因模型會將相同紋理或特徵物件進行辨識，因此畫有人臉並帶有口罩的蒙娜麗莎畫像被偵測到是可預期的，整體來說對偵測口罩仍有不錯效果。

## 結論

YOLO 是一個快速、準確的物件偵測系統，透過與 RCNN 不同的方式來處理物件偵測和分類，可以得到很快的速度及不錯的準確度，且隨著版本更新，有著不同網路架構和其他改動，其速度和正確率也隨之提升，相當適合用來做即時運算。本次專題嘗試實作 yolo，從準備資料到完成訓練並測試，本次雖說一開始遇到諸多阻礙，但仍然有訓練出有成果不錯並有實用型的模型，但相信在未來在有其他需求或研究時也能有很好的應用。

## 參考文獻

- [1] <https://medium.com/ching-i/yolo-c49f70241aa7>
- [2] <https://chih-sheng-huang821.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92->

- [%E7%89%A9%E4%BB%B6%E5%81%B5%E6%B8%AC-you-only-look-once-yolo-4fb9cf49453c](#)
- [ 3] [https://blog.csdn.net/weixin\\_38673554/article/details/105861016](https://blog.csdn.net/weixin_38673554/article/details/105861016)
- [ 4]<https://chtseng.wordpress.com/2018/09/01/%E5%BB%BA%E7%AB%8B%E8%87%AA%E5%B7%B1%E7%9A%84yolo%E8%BE%A8%E8%AD%98%E6%A8%A1%E5%9E%8B-%E4%BB%A5%E6%9F%91%E6%A9%98%E8%BE%A8%E8%AD%98%E7%82%BA%E4%BE%8B/>
- [ 5] <https://hackmd.io/@allen108108/r1-wSTAjs#%E7%B5%90%E8%AB%96-Conclusion>
- [ 6] 許碩芳, “一個基於 yolo 模型進行即時物件辨識之研究” 靜宜大學資訊傳播工程學系,碩士論文,2019
- [ 7]<https://medium.com/%E7%A8%8B%E5%BC%8F%E5%B7%A5%E4%BD%9C%E7%B4%A1/yolo-v2-%E7%89%A9%E4%BB%B6%E5%81%B5%E6%B8%AC-%E8%AB%96%E6%96%87%E6%95%B4%E7%90%86-a8e11d8b4409>