探索基於 YOLO 口罩外觀偵測模型在小型資料集下的效能提升表現方法
Exploring the Performance Improvement Method of YOLO-based Mask Appearance Detection Model under Small Data Sets

張倖瑜 [1] 何承遠 [2*]
[1] 亞洲大學資訊工程學系
[2] 國立臺灣大學資訊管理學系
Hsing-Yu Chang, Dept. of Computer Science and Information Engineering, Asia University
Cheng-Yuan Ho, Dept. of Information Management, National Taiwan University
*Email: tommyho@ntu.edu.tw

摘　要

COVID-19 於西元 2019 年底時爆發，並且迅速的影響至全球各地，各個國家為了防範此病毒，採取了多項措施，其中一種是使用醫療口罩防止病毒入侵，且由於口罩比起護目鏡和防護衣是較易取得且穿戴方便、防護力佳，使得使用量與日俱增。

為了讓工廠在大量製造時能快速的藉由外觀先初步辨認口罩的面罩和耳掛完整性，本研究利用物件偵測模型 YOLO 架構做口罩的外觀偵測，但因 YOLOV7 整體架構較大且利用 GPU 記憶體容量又大(於本研究實驗，最大 GPU 記憶體使用量為 5.79G)，在硬體上較不那麼高級的情況下很不方便訓練，且在做簡單分辨的外觀上有點殺雞焉用牛刀了，所以本研究最終使用了以 YOLOV7-tiny 為基準下去做訓練，也讓電腦的負擔下降許多(於本研究實驗，最大 GPU 記憶體使用量為 1.46G)，但在訓練時發現了準確度不夠高的問題。例如:訓練圖片總數量在最少 160 張、最多 750 張時，測試結果的準確度約為 28%至 95%。因此，本研究更改 YOLOV7-tiny 的內部架構，像是增加卷積層、池化層和跨層合作等，使模型在小型資料集下能提高準確度且維持原本的訓練、測試和驗證速度。在相同資料下(訓練圖片總數量在最少 160 張、最多 750 張時)，本研究方法的測試結果之準確度提高 5%至 32%，亦即準確度為 37%至 99%。

關鍵字：物件偵測、深度學習、口罩、小型資料集

一、序論—背景、動機和研究方法

由於在 2019 年年底時 COVID-19 的出現，影響了全世界，大家也透過媒體得知有研究表示戴口罩可以防止病毒，所以大家開始大量的購買口罩，甚至在台灣有一度供不應求，政府透過政策限制了每個人可以購買的數量，也同時防止了不孝的商人故意哄抬價格，但由於疫情的情況越發嚴重，大家的使用需求又增加了不少。

不管是醫療方面、服務業等等的接觸到的人都太過繁雜，甚至還有一天需要汰換掉很多片口罩的情況，且幾乎每一年都有流行感冒，口罩又被更大量的使用。

傳統的工廠或是較小型的工廠大多都使用人力去進行品質管控，可是時代慢慢的在演變，有很多的工廠都有配合人工智慧等方法進行品質管控，一方面可以減少人力資源，也可以加快在生產線上的速度，雖然以前期來說可能需要花費一些機器等設備的費用，但是以長期來看，對於工廠來說還是省掉很多成本與時間。

口罩因為疫情及流行感冒的關係被大量使用，製作口罩的工廠一定日以繼夜的不斷生產口罩，但是在生產過程中不免俗的一定會有一些壞掉不能販售出去的口罩，可是若是以人力下去一個一個檢查，又花時間又花金錢，甚至還會有因為疲勞看錯或是不小心將好與壞的口罩分類錯誤的情況，為了避免這種情況的發生，希望在口罩大量製造的過程中可以快速的從外觀上初步的先辨認出口罩的耳掛是否完整，也可以讓公司了解到自家工廠出錯的部分並加以更正。

本研究更改 YOLOV7-tiny 模型的架構，更改後的模型取名為 Reorganized YOLOV7-tiny Model，與原 YOLOV7-tiny 模型比較準確度、速度、時間，透過實驗了解 Reorganized YOLOV7-tiny Model 是否比 YOLOV7-tiny 好，有優化效果，實驗步驟如下：

第一步：建立口罩資料集，將口罩進行分類，利用固定的攝影機進行拍照及存檔。

第二步：資料前處理，將影像透過 MAKESENSE(影像標記網站)標記影像的類別及位置並儲存，且輸出 YOLO 格式的文字檔。

第三步：拆資料，照片都處理完成後，拆成三個資料集，Training Data、Validation Data、Test Data。

第四步：模型訓練&驗證，YOLOV7-tiny 模型及 Reorganized YOLOV7-tiny Model 都丟入 Training Data 與 Validation Data 資料集及 YOLO 格式的文字檔進行模型訓練，訓練完成後將模型儲存，並使用 Test Data 進行模型的驗證，並儲存其準確度。

模型訓練過程及驗證過程的流程圖如圖 1 及圖 2 所示。圖 1 先將取得的口罩影像進行分類，分類成 Training Data、Validation Data、Test Data 三組，接著並將 Training Data 和 Validation Data 丟入

* 第四屆台灣軟體工程研討會(論文集, Volume 3)
畫經費補助

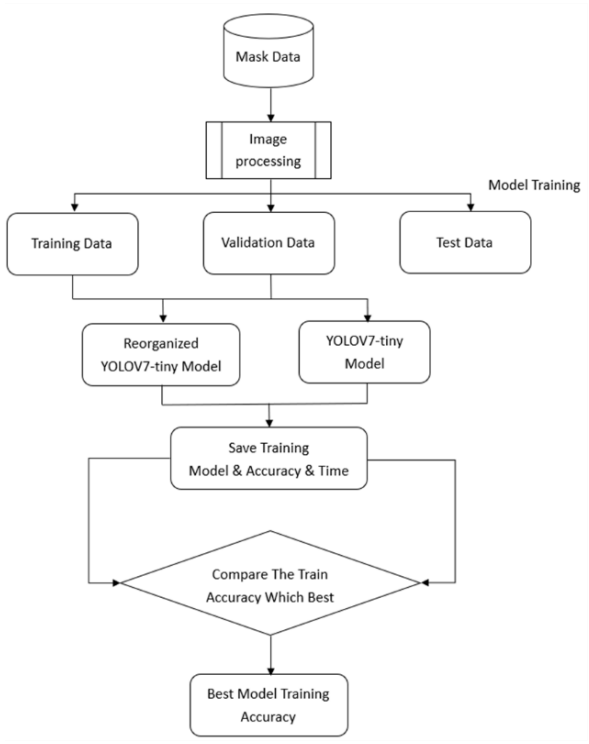

圖 1 模型訓練過程

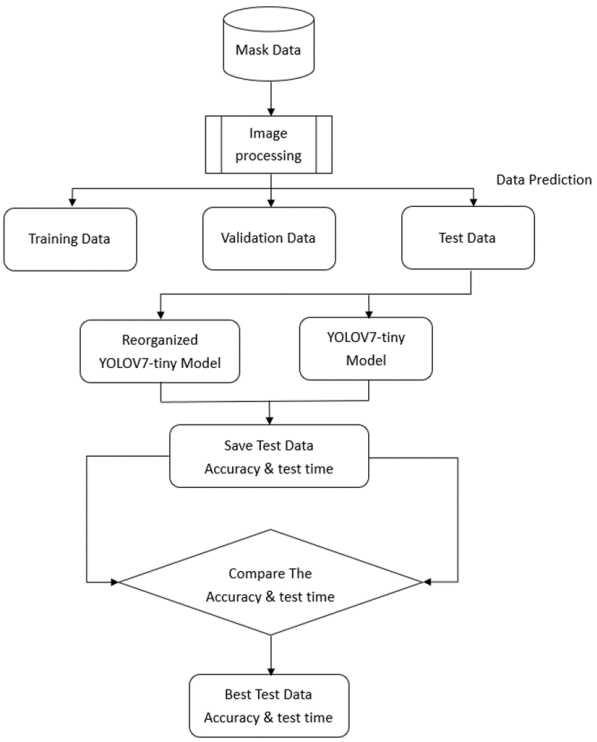

圖 2 模型驗證過程

YOLOV7-tiny 模型及 Reorganized YOLOV7-tiny Model 做訓練後並儲存訓練好的模型、準確度及速度，並將 YOLOV7-tiny 模型及 Reorganized YOLOV7-tiny Model 兩者去做準確度比較，並記錄下來。圖 2 則是將取得的口罩影像進行分類後分為三類(Training Data、Validation Data、Test Data)中的 Test Data 丟進去訓練好的 YOLOV7-tiny 模型及 Reorganized YOLOV7-tiny Model 去做測試，並儲存測試後的準確度及時間，並比較訓練好的 YOLOV7-tiny 模型及 Reorganized YOLOV7-tiny Model 兩者的準確度及時間並儲存。

本研究目的為在小型資料集下口罩外觀偵測模型效能提升的方法。第二節介紹相關文獻，第三節介紹本研究的方法、環境及資料集。第四節描述模型的設計方法與詳細架構圖，接下來，在第五節整理實驗後的結果、驗證、測試及比較，最後的第六節總結本次研究、探討遇到的問題及未來的研究方向。

二、文獻回顧

在一張圖像之中可以把物體框出來並且知道它的類別及機率的技術稱為物件偵測，而物件偵測有分成兩大類別。第一類為 One stage，第二類為 Two stage，前者是可以一次得到物體的類別及位置的，例如 YOLO (You Only Look Once) [1]、EfficientDet [2]，而後者則是先框出物件的位置再去辨認類別，例如 R-CNN[3]、Fast R-CNN[4]、Faster R-CNN[5]。

YOLO 是物件偵測的其中一個技術，為一整個系列，從 YOLOV1 開始一路到 YOLOV7，從 2015 年到最新的 2022 年，YOLOV7 是截至 2022 年為止在準確度上和速度上都超越目前的其他物件偵測工具 [1]。

YOLOV7 優化的地方有:

1. Model re-parameterized

Model re-parameterized 分成兩種，一種是 model-level，另一種是 module-level，model-level 會訓練同一個模型但使用很多不同的訓練資料後將這些模型的權重進行平均，或是訓練同一個模型但使用不同迭代次數的權重進行加權平均。而 module-level 會在訓練時把 module 拆成很多個模塊分支，並在推理時把分支合成等價的模塊。

2. Model scaling

Model scaling 中較常見的因子有 Resolution (size of input image)、Stage(number of feature pyramid)、Depth (number of layer)、Width (number of channel)，發現如果是 concatenation-based 架構，後面的 layer 會被前面的縮放影響，導致使用率下降，所以作者提出 compound model scaling，當增加 Depth 時，會計算出輸出通道的變化量，並以相同變化量對 Width 進行放大。

3. Coarse for auxiliary and fine for lead loss

在模型中加入 auxiliary head 可以讓模型收斂得更好，增加整體的性能。目前比較常見的方法 lead head 和 auxiliary head 分開進行 label assignment，而 YOLOV7 作者提出了兩種 label assignment。兩者都是以 lead head 去做主軸，前者的話因為 lead head 學習能力較好，所以讓 auxiliary head 去學習 lead head 已學到的資訊後，就可以讓 lead head 更專注在其他沒學習到的資訊；後者則是因 auxiliary head 學習能力較弱，透過 coarse label，避免

漏掉重要的資訊，透過 Dynamic label assignment strategy 使得訓練更加完整，優化訓練的過程。

文獻[6]利用 CNN 為基礎優化架構進行影像辨識，主要是辨識人配戴時的情況。研究內容描述目前大多的文獻內容的一般口罩辨識，都是以素的口罩為主，但是現在越來越多不同的花樣及款式，所以作者希望讓模型也可以辨識有花色的口罩，同時提高準確度。文章內有做兩種實驗，一種是訓練資料分成有配戴口罩跟沒有配戴後，做資料增強後進行 VGG16 模型訓練得到的 CNN 模型與原本的 VGG16 做比較；另一種是有先經過人臉識別的分類器模型先將較遠景的影像擷取臉部後輸出，在進行前一種的所有步驟。最後有跟 SSD、YOLOV3、YOLOV4、YOLOV5 做比較，數據顯示 YOLOV4 的準確度跟該研究的結果一樣，但召回率較低且 FN 數量比 YOLOV4 少。

本研究與上述文獻[6]研究主要的區別有：第一，本研究主要是做在工廠供應鏈上的口罩辨識，而文獻[6]主要是在指人們配戴時的辨識；第二，本研究的口罩都是素面的口罩，而文獻[6]主要是以多樣花色進行研究；第三，本研究改的是 YOLOV7-tiny 的架構，文獻[6]主要是改 VGG16 的架構。

同樣是 2022 年的文獻[7]利用 ResNet152V2 [8]、InceptionV3 [9]、Xception [10]模型並測試搭配影像預處理是否有差異等，其研究內容主要是在做口罩在產線上的瑕疵，且共分為 5 種瑕疵，裁切瑕疵、邊條斷裂、邊條過長、掛繩斷裂、沒有掛繩。本研究與文獻[7]主要的區別在於：第一，本研究有做架構上的更改，而文獻[7]利用目前現有的模型；第二，本研究並未做影像的灰階化和背景濾除，而文獻[7]特地做灰階化和背景濾除；第三，本研究主要在做外觀辨識，而文獻[7]著重在於瑕疵的辨識。

文獻[11]也同樣是做口罩缺陷檢測，研究內容主要採用多種 CNN 模型進行檢測，如 Faster R-CNN、YOLOV3、YOLOV5 等。本研究與文獻[11]主要的區別如後：第一，本研究有做架構上的更改，而文獻[11]是利用目前現有的模型；第二，本研究主要在做外觀辨識，而文獻[11]著重在於瑕疵的辨識；第三，本研究並未做影像的灰階化，文獻[11]有做灰階化處理，且有做輪廓提取並骨架化。

三、研究方法

(一) 實驗環境介紹

本研究使用的硬體有桌上型電腦(設備及作業系統如表 1)、小型工業流水線(圖 3)、運動攝影機(型號為 insta 360 one R，圖 4)及小型照明燈(圖 5)來取得工業流水線在運作時的口罩影像。

用來建模和預測的桌上型電腦硬體配置及環境介紹如表 2。由於 YOLOV7 模型需要的系統和套件在版本有其限制，所以必須要在乾淨的環境下，如透過 Anaconda 另創新虛擬環境並安裝必要套件，例如 Numpy、Opencv-python、pandas、torch、tensorboard 等。如需要正確的版本規格，請參考 YOLOV7 作者給予的 GitHub 網站。

表 1 取得口罩影像所用硬體及作業系統

硬體配置	CPU 處理器	Intel i5-9400F
	RAM 記憶體	32GB
	GPU 顯示卡	Nvidia RTX3060*2
系統配置	作業系統	Ubuntu 20.04

表 2 建模所用硬體配置及環境介紹表

硬體配置	CPU 處理器	12th Gen Intel(R) Core(TM)i5-12400 2.5GHz
	RAM 記憶體	16 GB
	GPU 顯示卡	Nvidia RTX3070
系統/軟體/套件配置	作業系統	Window 10
	CUDA	11.4
	Python	3.9.13
	Numpy	1.23.3
	Opencv-python	4.6.0.66
	pandas	1.5.0

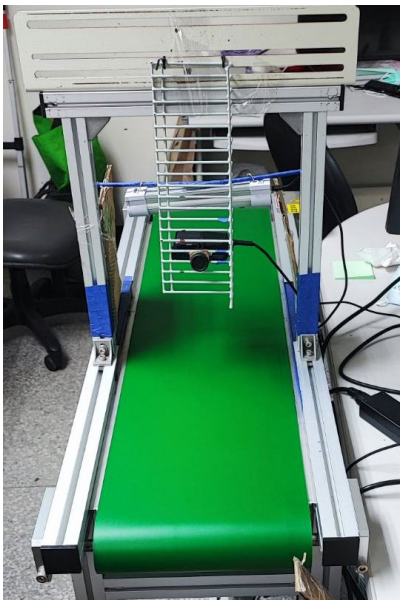

圖 3 實驗室建立的小型工業流水線

圖 4 運動攝影機(型號為 insta 360 one R)

圖 5 小型照明燈

(二) 資料集介紹

本研究所用資料集共分成三大類：第一類為正常，意即面罩完整且耳掛也完整(圖 6)、第二類為兩邊皆完全沒有耳掛(圖 7)或是僅有一側沒有耳掛，另一側耳掛為正常(圖 8)、第三類為耳掛斷掉，包含兩側耳掛皆斷掉(圖 9)或是僅有一側耳掛斷掉(圖 10)。不把第二類與第三類合併為同一類的最主要原因是第三類是一開始有將耳掛接上去，但是因為熔點沒有做好，導致耳掛斷掉的情形(如圖 9、圖 10 中粉色圓圈框出的位置)，但第二類是一開始一邊或是兩邊沒有接上耳掛，並非是熔點造成的問題。

圖 6 第一類(正常)

圖 7 第二類(兩邊皆完全沒有耳掛)

圖 8 第二類(一側沒有耳掛、另一側耳掛正常)

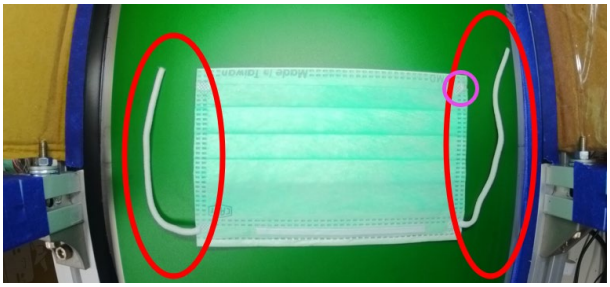

圖 9 第三類(兩側耳掛皆斷掉)

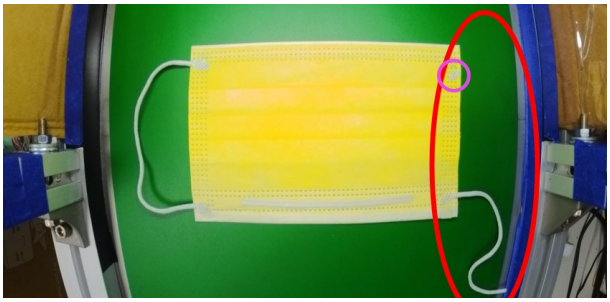

圖 10 第三類(僅有一側耳掛斷掉)

(三) 資料集處理

藉由運動攝影機所取得的口罩影像使用影像標記網站(MAKESENSE)標記出影像的類別和位置，標記完成的影像如圖 11。標註的框為邊界框(bounding box，簡寫 bbox)，並輸出 YOLO 格式的文字檔(圖 12)，文字檔內的數字由左到右分別為類別 ID、X、Y(分別代表框出的 bbox 中心位置與圖像高寬的比值，中心座標 X 和 Y)、W、H(分別代表框出的 bbox 高寬與圖像高寬的比值，寬和高)。YOLO 座標的標示圖如圖 13，至於 bbox 座標的標示及 YOLO 格式的計算方式比照可參考圖 14。

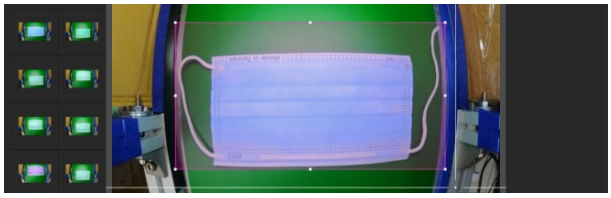

圖 11 標記完成的影像

圖 12 YOLO 格式文字檔

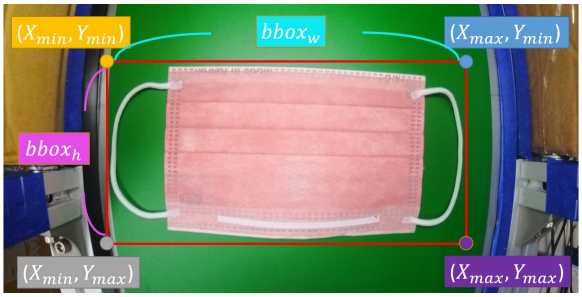

圖 13 YOLO 座標標示

X = (Xmin+ Xmax) / 2w ， Y = (Ymin + Ymax) / 2h

W = (Xmax – Xmin) / w ， H = (Ymax – Ymin) / h

(算式中的w, h 是原本圖像的寬與高)

圖 14 YOLO 格式計算

另外，為了瞭解小型資料集對 YOLO 模型和本研究提出的模型影響，本研究將主資料集拆分成 12 種不同數量的子資料集，表 3 分別秀出訓練資料集和驗證資料集的詳細數據和數量。

四、模型設計

(一) YOLOV7-tiny 架構介紹

YOLOV7-tiny 的架構如圖 15 所示，架構分成兩大區塊，第一區塊為 Backbone、第二區塊為 Head，分別如圖 15 中紅線左半部分和右半部分。程式架構的呈現如圖 16 (Backbone)和 17 (Head)所示，圖 16 和 17 裡有很多中括號，其組成為[from, number, module, args]。from 代表輸入的是哪一層，最常見的為-1，代表上一層，也可以直接打上數字，例如 10，表示從第 10 層獲取，number 代表需要多少個

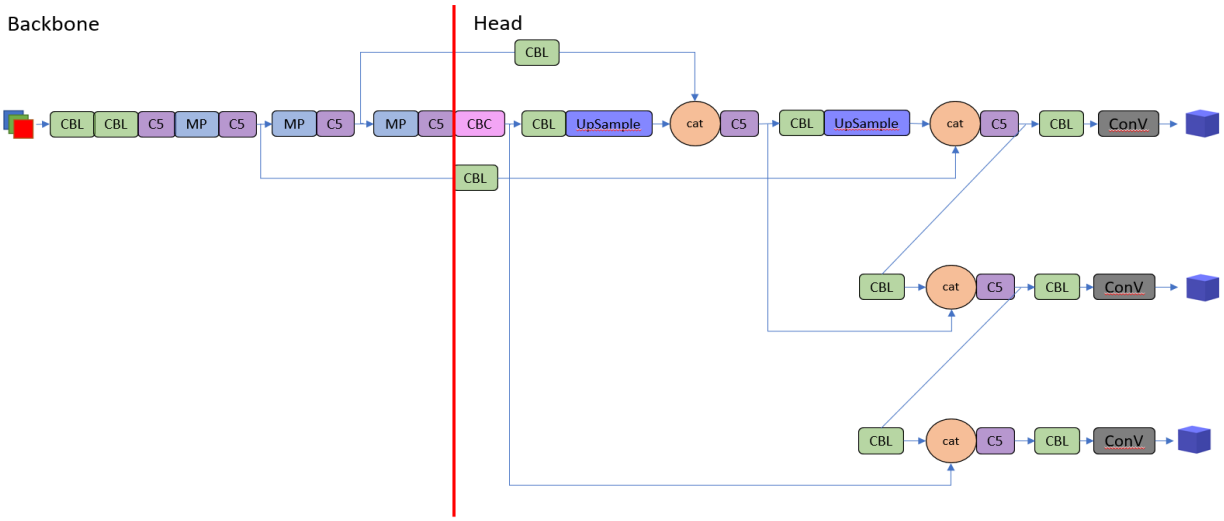

圖 15 YOLOV7-tiny 架構

表 3 訓練、驗證資料集詳細數據和數量

類別 / 序號	第一類 (正常) (張)	第二類 (兩邊皆完全沒有耳掛或僅有一側沒有耳掛) (張)	第三類 (兩側耳掛皆斷掉或僅有一側耳掛斷掉) (張)	資料總數 (張)
TRAINING DATA				
1	250	250	250	750
2	200	200	160	560
3	200	160	200	560
4	160	200	200	560
5	200	160	160	520
6	160	200	160	520
7	160	160	200	520
8	160	160	160	480
9	160	80	80	320
10	80	160	80	320
11	80	80	160	320
12	80	40	40	160
VALIDATION DATA				
1	63	63	63	189
2	50	50	40	140
3	50	40	50	140
4	40	50	50	140
5	50	40	40	130
6	40	50	40	130
7	40	40	50	130
8	40	40	40	120
9	40	20	20	80
10	20	40	20	80
11	20	20	40	80
12	20	10	10	40

一樣模塊組成，module 代表模塊的種類，args 代表不同的引數。更言之，Backbone 有三個模塊，分別為 CBL (Convolution + Batch Normalization + LeakyRelu，功能為取得特徵，圖 18)、C5 (功能為整合多個特徵圖，提高準確性，圖 19) 和 MP (Max Pooling，功能是讓模型不要學太多邊緣或不重要特徵，以免模型太敏感，圖 20)；Head 有兩個模塊，分別為 CBC (為取得不同大小的特徵圖，提高整體效能及準確度，圖 21)，UpSample 的採樣方式是以 Nearest Neighbor Interpolation 方法實施，其功能是放大影像。

```
# yolov7-tiny backbone
backbone:
  # [from, number, module, args] c2, k=1, s=1, p=None, g=1, act=True
  [[-1, 1, Conv, [32, 3, 2, None, 1, nn.LeakyReLU(0.1)]],  # 0-P1/2

   [-1, 1, Conv, [64, 3, 2, None, 1, nn.LeakyReLU(0.1)]],  # 1-P2/4

   [-1, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 2
   [-2, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 3
   [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],  # 4
   [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],  # 5
   [[-1, -2, -3, -4], 1, Concat, [1]],                     # 6
   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 7

   [-1, 1, MP, []],  # 8-P3/8
   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [[-1, -2, -3, -4], 1, Concat, [1]],
   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 14

   [-1, 1, MP, []],  # 15-P4/16
   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-2, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [[-1, -2, -3, -4], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 21

   [-1, 1, MP, []],  # 22-P5/32
   [-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-2, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
   [[-1, -2, -3, -4], 1, Concat, [1]],
   [-1, 1, Conv, [512, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 28
  ]
```

圖 16 以程式碼呈現的 Backbone

```
# yolov7-tiny head
head:
  [[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-2, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [-1, 1, SP, [5]],
   [-2, 1, SP, [9]],
   [-3, 1, SP, [13]],
   [[-1, -2, -3, -4], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
   [[-1, -7], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 37

   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 38
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],              # 39
   [21, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 40 route backbone P4
   [[-1, -2], 1, Concat, [1]],                              # 41

   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 42
   [-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 43
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],  # 44
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],  # 45
   [[-1, -2, -3, -4], 1, Concat, [1]],                     # 46
   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 47

   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 48
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],            # 49
   [14, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 50 route backbone P3
   [[-1, -2], 1, Concat, [1]],                            # 51

   [-1, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 52
   [-2, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 53
   [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 54
   [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 55
   [[-1, -2, -3, -4], 1, Concat, [1]],                    # 56
   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 57

   [-1, 1, Conv, [128, 3, 2, None, 1, nn.LeakyReLU(0.1)]], # 58
   [[-1, 47], 1, Concat, [1]],                             # 59

   [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 60
   [-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 61
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 62
   [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 63
   [[-1, -2, -3, -4], 1, Concat, [1]],                    # 64
   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 65

   [-1, 1, Conv, [256, 3, 2, None, 1, nn.LeakyReLU(0.1)]], # 66
   [[-1, 37], 1, Concat, [1]],                             # 67

   [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 68
   [-2, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 69
   [-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 70
   [-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 71
   [[-1, -2, -3, -4], 1, Concat, [1]],                     # 72
   [-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 73

   [57, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 74
   [65, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 75
   [73, 1, Conv, [512, 3, 1, None, 1, nn.LeakyReLU(0.1)]], # 76

   [[74,75,76], 1, IDetect, [nc, anchors]],   # Detect(P3, P4, P5)
  ]
```

圖 17 以程式碼呈現的 Head

圖 18 CBL 模塊

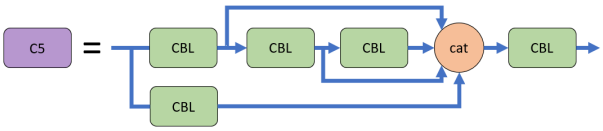

圖 19 C5 模塊

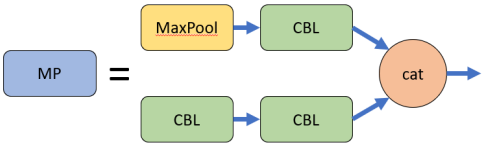

圖 20 MP 模塊

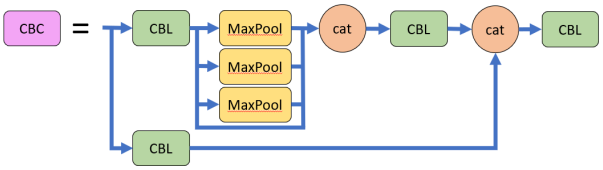

圖 21 CBC 模塊

(二) Reorganized YOLOV7-tiny 架構設計理念

在小資料集或資料不足的情況下，本研究發現 YOLOV7 和 YOLOV7-tiny 模型的準確度有很大的研究空間(詳細數據請參閱下一節)，因而期望能發展出一個介於 YOLOV7 和 YOLOV7-tiny 的模型架構，卻又能比前二者的模型準確度要高，意即該模型架構在使用軟硬體資源以及花費時間建模時，能跟 YOLOV7-tiny 差不多，同時該模型架構的複雜度不會如 YOLOV7 那般複雜和多層，但該模型架構可提高效能並得到高準確度。

本研究為了讓模型能更減少邊緣資料的敏感程度，同時能學到更多的影像特徵，在 Backbone 中多加入一組 MP＋C5。另一方面，為讓模型能取得不同大小的特徵圖，提高整體效能及準確度，本研究將 Head 中的 CBC 換成 SPPCSPC(Spatial Pyramid Pooling + Cross-Stage-Partial + Convolution)，其模塊如圖 22。進一步而言，SPPCSPC 的內部模組是使用 CBS (Convolution + Batch Normalization +Silu，功能為取得特徵，如圖 23)。CBS 和 CBL 的不同，在於 Silu 和 LeakyRelu 這兩個激活函數，前者所得準確度會高於後者，但兩者的計算速率並無太大差別。另外，SPPCSPC 比起 CBC 要多了 3 次取得特徵，且其中有 2 個是在 MaxPool 前，其優點是可以透過不一樣大小的 MaxPool 去更好學習小的特徵點跟大的特徵點。經過加入和置換後，圖 24 顯示 Reorganized YOLOV7-tiny 模型的完整架構圖。

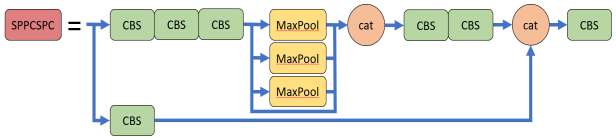

圖 22 SPPCSPC 模塊

圖 23 CBS 模塊

五、實驗結果與分析

如表 3 所示，本研究的訓練資料和驗證資料分別各有 12 種資料集。實驗採用相同序號來做為同一組配對，意即使用第 N 序號的訓練資料來建模，將採用第 N 序號的驗證資料來求得該訓練模型(或稱預測模型)來驗證其準確度，其中$1 \le N \le 12$。

(一) YOLOV7-tiny 模型驗證

YOLOV7-tiny 模型平均訓練時間與其驗證資料準確度結果呈現在表 4。從表 4 可發現，在資料集數量為 480 張以上時，YOLOV7-tiny 準確度至少都有 90%，甚至可到 97.6%。另一方面，訓練時間大約都在 74 分鐘至 120 分鐘左右。然而，在訓練資料集個數不大於 360 張，可以發現準確度明顯下降，且下降幅度非常劇烈，從 73%劇降到 21.6%。

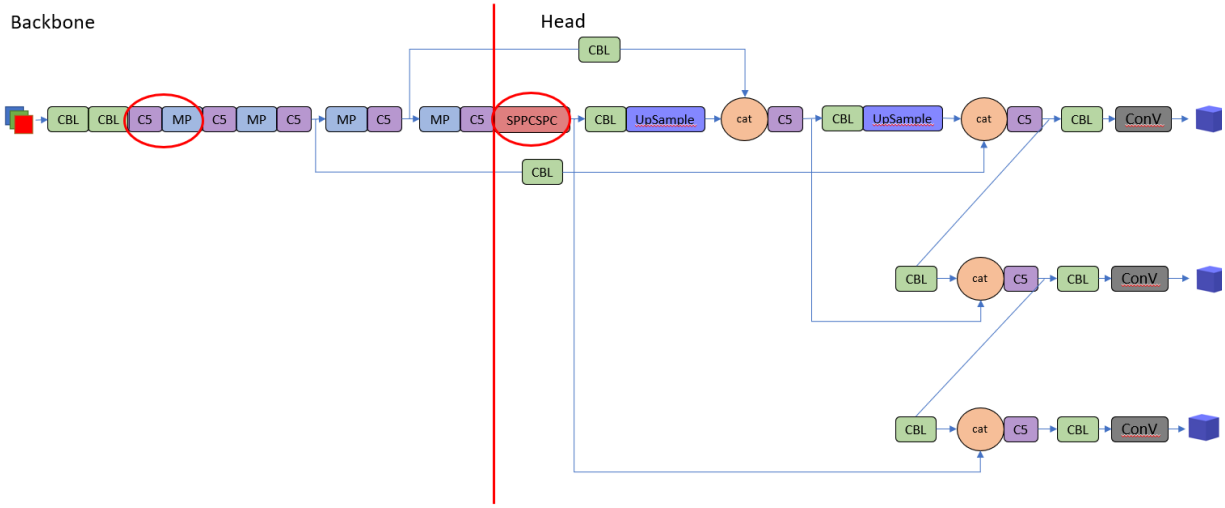

圖 24 Reorganized YOLOV7-tiny 架構

(二) Reorganized YOLOV7-tiny 模型驗證

Reorganized YOLOV7-tiny 採用相同的訓練資料、驗證資料和實驗步驟後，所得平均訓練時間與驗證資料準確度結果呈現在表 5。從表 5 可以發現，在資料集數量為 480 張以上時，本研究提出的模型準確度都在 93.6%以上，甚至可到 98.6%，整體平均為 96.10%。另外，Reorganized YOLOV7-tiny 訓練時間都小於兩小時，最久為 119 分鐘，最短為 75 分鐘，其所使用的訓練時間跟 YOLOV7-tiny 幾乎相近。在訓練資料集個數不大於 360 張時，可發現 Reorganized YOLOV7-tiny 的效能比 YOLOV7-tiny 要來得高。如在 320 張，Reorganized YOLOV7-tiny 的準確度皆在 70%以上，甚至可達 89.60%，是 YOLOV7-tiny 的 1.06 倍到 1.27 倍之多。

表 4 YOLOV7-tiny 模型訓練時間與準確度

序號	訓練資料總數(張)	訓練時間(小時)	驗證集準確度
1	750	2.016	97.60%
2	560	1.384	96.30%
3	560	1.471	93.60%
4	560	1.405	91%
5	520	1.341	97%
6	520	1.336	95.60%
7	520	1.356	94%
8	480	1.229	90.60%
9	320	0.819	73%
10	320	0.82	68.60%
11	320	0.826	55.30%
12	160	0.413	21.60%

表 5 本研究提出的模型訓練時間與準確度

序號	訓練資料總數(張)	訓練時間(小時)	驗證集準確度
1	750	1.986	98%
2	560	1.458	98.60%
3	560	1.44	93.60%
4	560	1.387	95%
5	520	1.378	96%
6	520	1.363	96%
7	520	1.348	97%
8	480	1.253	94.60%
9	320	0.845	89.60%
10	320	0.823	73.30%
11	320	0.845	70.60%
12	160	0.401	23.60%

(三) 兩模型建模時間和驗證資料準確度比較

表 6 整理上述兩模型的平均建模時間和其驗證資料準確度。透過表 6 統整，可了解在這 12 種資料集下，從訓練時數方面而言，雖然 Reorganized YOLOV7-tiny 偶爾花費的時間較多(約多 2 分鐘)，但其準確度皆高於原本的 YOLOV7-tiny。

另外，從實驗中可以發現在資料集數量同為 320 張時(表 3 序號 9~11)，口罩影像在三類別資料比例為 2:1:1 的時候，是兩模型準確度較高的時候；在其他比例時，兩模型的準確度還有進步的空間。然而，在資料集數量更少時，如 160 張，即使三類別資料比例是 2:1:1，兩模型的準確度會驟降至不到 25%。由此實驗可知，在可接受的準確度前提下，Reorganized YOLOV7-tiny 和 YOLOV7-tiny 對於小資料集的資料數量的可能極限為 320 張。

表 6 兩模型驗證準確度與訓練時間平均比較表

訓練資料集總和數量 (張)	Reorganized YOLOV7-tiny 準確度/訓練時間(小時)	YOLOV7-tiny 準確度/訓練時間(小時)
750	98.0% / 1.986	97.6% / 2.016
560	95.7% / 1.428	93.6% / 1.42
520	96.3% / 1.363	95.5% / 1.344
480	94.6% / 1.253	90.6% / 1.229
320	77.8% / 0.837	65.6% / 0.821
160	23.6% / 0.401	21.6% / 0.413

(四) 兩模型的實務測試資料準確度比較

隨機抽取 105 張未曾在訓練資料和驗證資料出現的測試圖片(三類的數量分別是 65 張、20 張和 20 張)，做為兩模型的實務測試資料。實際的操作為 Reorganized YOLOV7-tiny 和 YOLOV7-tiny 各有 12 個模型(經表 3 的 12 個子資料集建模)，餵入此 105 張測試圖片資料，並量測其最終準確度，其結果可參閱表 7 和表 8。

從表 7 和表 8，可發現在超過 320 張訓練資料所建立出的模型(序號 1~8)，無論是以 Reorganized YOLOV7-tiny 還是 YOLOV7-tiny 為架構，其準確度皆是可實用於現實生活和產線上的。更進一步，Reorganized YOLOV7-tiny 的準確度落在 92.30%到 99.00%之間，而 YOLOV7-tiny 的數據落在 88.50%到 95.2%之間，本研究所提的模型較 YOLOV7-tiny 表現優異。

另一方面，當是以小資料集訓練出來的模型，Reorganized YOLOV7-tiny 在準確度的表現絕大多數模型比 YOLOV7-tiny 的模型要來得好，最高可達 1.3 倍的效果。

表 7 Reorganized YOLOV7-tiny 測試結果

序號	辨識正確的數量(張)	辨識錯誤的數量(張)	準確度
1	104	1	99.00%
2	103	2	98.00%
3	99	6	94.20%
4	97	8	92.30%
5	98	7	93.30%
6	101	4	96.10%
7	97	8	92.30%
8	98	7	93.30%
9	81	24	77.10%
10	75	30	71.40%
11	70	35	66.60%
12	39	66	37.10%

表 8 YOLOV7-tiny 測試結果

序號	辨識正確的數量(張)	辨識錯誤的數量(張)	準確度
1	100	5	95.20%
2	100	5	95.20%
3	97	8	92.30%
4	97	8	92.30%
5	95	10	90.40%
6	100	5	95.20%
7	93	12	88.50%
8	100	5	95.20%
9	75	30	71.40%
10	76	29	72.30%
11	56	49	53.30%
12	30	75	28.50%

六、結論與建議

此次研究將模型增加一些層數，使模型訓練特徵時更完整，且在小資料集或資料集不足的情況下，依然可以維持一定的準確度，增加初步分類時的準確度。同時，可以了解到像口罩這種較簡單的影像，不需要用到很強大的模型，就可得到很不錯的成效。

Reorganized YOLOV7-tiny Model 在資料集真的很缺乏時(約 320 張)時，還是能達到 70%左右的準確度，而在足夠的情況下(約 750 張)時去訓練後的模型，測試結果將整體準確度提高到幾乎 99%的結果。

透過這次研究實驗得知像口罩這種較簡單的影像訓練配上 YOLOV7-tiny 這種類型的小型架構時，大約要有多少資料集數量與硬體設備，由於瞭解資料不需要很多，所以可以減少收集資料集的時間，還能減少標記影像的時間，更能減少訓練時龐大的時間。

然而，這次研究使用的攝影機並非很高階，在供應鏈運輸速度較快時，影像會變得極度模糊導致模型訓練結果很差，但在一定的速度下還是可以很清晰的取得影像。在實驗過程中，發現此次使用的攝影機也有些微的廣角功能，可能會造成模型部分偏差，希望在設備更完善時，可以有更好訓練成果。

未來的研究方向有幾種，可以再接下去研究如何在資料集不足的情況下維持模型的準確度；也可以試著在很多不同種類的的資料集下，嘗試建立一種模型都能很好的適應，不管是在準確度上還是在時間上都有不錯的成效；也或許可以思考當供應鏈上的運輸速度很快時，影像很模糊的情況下，如何辨識影像的類別，除了更換解析度更高的攝影機外是否能有更好的模型來進行處理或是改成影片的方式訓練。另外，當預計辨識的影像有重疊嚴重的情況時，如何完整辨識影像類別也是一種可能的研究方向。

參考文獻

[1] Chien-Yao Wang, Alexey Bochkovskiy, Hong-Yuan Mark Liao (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696.

[2] Mingxing Tan, Ruoming Pang, Quoc V. Le (2019). EfficientDet: Scalable and Efficient Object Detection. arXiv preprint arXiv:1911.09070.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv preprint arXiv:1311.2524.

[4] Ross Girshick (2015). Fast R-CNN. arXiv preprint arXiv:1504.08083.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv preprint arXiv:1506.01497.

[6] Ting-Guang Jiang (2022). The patterned face mask detection. https://hdl.handle.net/11296/bas7n2.

[7] Ting-Han Wang (2022). Mask inspection management systems. https://hdl.handle.net/11296/v36z65.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015). Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna (2015). Rethinking the Inception Architecture for Computer Vision. arXiv preprint arXiv:1512.00567.

[10] François Chollet (2016). Xception: Deep Learning with Depthwise Separable Convolutions. arXiv preprint arXiv:1610.02357.

[11] LIANG, CHIA-WEI (2022). Study on Defect Detection of Medical Mask: Combination of Deep Learning and Machine Vision. https://hdl.handle.net/11296/gx3nw5.