國立臺中科技大學
資訊管理系

人工智慧影像辨識
未戴安全帽辨識系統
Artificial Intelligence Image Recognition System for Detecting Missing Safety Helmets

智慧運算創新應用－E10

班　　級：資管五甲

指導老師：李銘峰 老師

專題組員：1110931043 黃國華

1110931036 張荃宇

1110931034 張倫鈞

1110931045 杜沅瑾

中 華 民 國 １１３ 年 １１ 月 ８ 日

目錄

表目錄

圖目錄

第一章 前言

1.1 研究背景與動機

在交通部 168 交通安全入口網資料公布騎乘機車未戴安全帽之致死率是有正確配戴者之 8.63 倍[1]。機車騎士及乘客配戴安全帽能降低事故死亡之風險，可知此一舉動之重要性與對生命安全的保障。自民國 86 年（下同）立法規定機車駕駛人或附載座人須戴安全帽並處以違者 500 元罰鍰，而根據警政署統計 92 年到 112 年安全帽違規取締件數（圖 1-1-1），確實大幅改善[2]。但有部分民眾仍抱持僥倖心態、貪圖一時方便而違規，進而造成嚴重事故傷害。

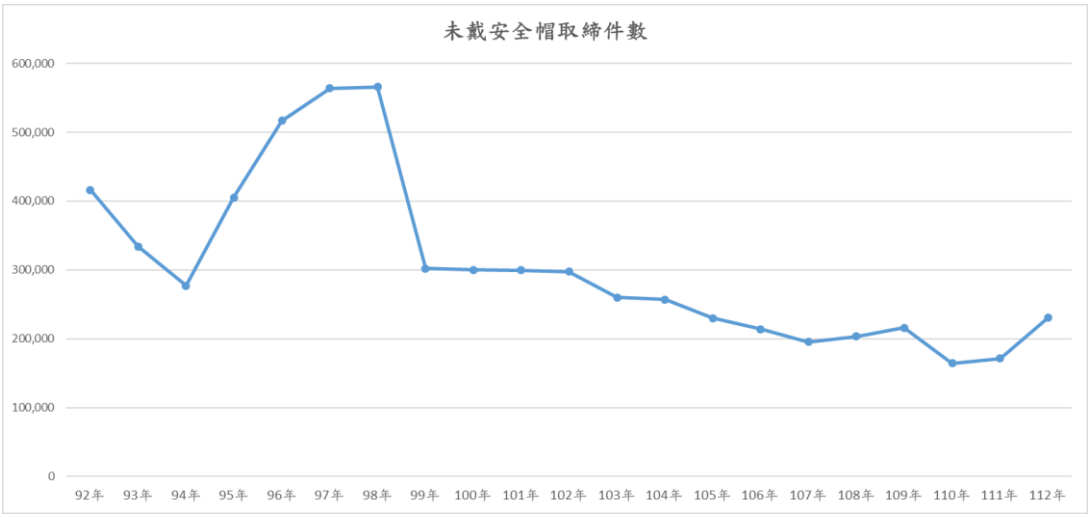

圖 1-1-1 未戴安全帽取締件數[2]

目前實施科技執法有六大項目：路口多功能執法、區間測速、違規停車、高架道路科技執法、高速公路出口匝道、聲音照相[3]。現行之科技執法項目並未包含安全帽取締，而其違規行為有賴警方人力與民眾檢舉監督，但此違規行為所造成的受傷或死亡數，尤以部分鄉鎮區域較為嚴重。據 112 年 1 月至 11 月的統計，未戴安全帽死傷人數之縣市排名，以臺南市、屏東縣、嘉義縣較多[4]。臺灣預計 113 年 6 月再度開放未戴安全帽檢舉項目[5]。且民眾設置或配戴行車紀錄器日益增加，以行車紀錄器和路口監視器等畫面，透過影像辨識技術偵測並舉發違法者。民眾之安全意識主動舉發與科技執法之增設，將可降低執法人力負擔，改正未戴安全帽之陋習。

因此本組研究如何利用影像辨識偵測機車騎士及乘客是否配戴安全帽，以供民眾自行檢舉該違規行為和日後科技執法的增設之參考。

1.2 研究目的

根據研究動機，本組希望以影像辨識舉發未戴安全帽之違規行為。一影像是由多個連續幀（frames，或稱影格）所構成，其中的一幀可能會出現任何類別的物體，首先要辨識該幀所包含之機車騎士和乘客，接著判斷其是否依規定配戴安全帽，若違規事實成立須記錄當下幀並識別其車牌以供開單舉發之依據，如此針對影像所有的幀進行處理。

綜合上述所言，整理以下三點本組影像辨識之目標：

1. 辨識機車駕駛人及乘客
2. 辨識該行為人未戴安全帽
3. 辨識該駕駛或乘坐之車輛之車牌號碼

而影像的來源主要是行車紀錄器、路口監視器畫面，本組之辨識須以此類影像進行即時處理，並滿足辨識目標。

1.3 市場應用分析

臺灣目前市場上針對機車安全帽辨識之產品較無相關資料，因此本組就其他區域及安全帽影像識別之相關應用進行探討。

中國大陸透過電子警察科技執法取締違規，以下以保亭黎族苗族自治縣之案例為例：保亭黎族苗族自治縣的民眾日常以摩托車、電動自行車為主要交通工具，但根據往年統計之交通事故中使用該工具而未戴安全帽死亡比例居高，當局警力有限且民眾安全意識低使該地無法有效改善此一現象[6]。為解決上述問題，當局政府實施「隨手拍與電子警察」的違法查處制度，除了開放民眾拍照舉發違規行為，政府也建置識別摩托車、電動車和騎乘人員未戴安全帽之模型，並用高像素監視器具 24 小時全天偵測對違規者進行裁處，電子警察設置後已開罰 6427 件違規。由違法糾正再到社群媒體與教育宣導等措施，使該地安全帽配戴率從 71.2%

上升至 95%以上，躍居全省第一[6]。中國於多個區域導入電子警察取締未依規定戴安全帽同樣頗具成效。

安全帽影像辨識也常運用於工地安全，以清波實業股份有限公司為例，該公司推出雲端(On-line)與落地版(Off-line)工地安全 AI 影像識別方案，工人未戴安全帽時可透過 Line notify 等平臺發送安全告警，除了快速、彈性且低成本投入部署，AI 上線後能針對現場環境與安全帽等特性之不同修正模型以提高正確率，藉此保障工地作業安全[7]。

綜合上述兩者市場實際案例，中國大陸運用監視器結合機車及安全帽的影像辨識彌補警力需求並確實落實執法，清波實業以工地安全為應用實現未戴工地安全帽之告警系統，兩者所採用之辨識影像為定點式的監視畫面，本組所開發之辨識系統除了定點的監視畫面，也要處理變動的行車紀錄畫面，相較於兩案例，在影像處理上會更為複雜。工地安全帽的辨識其安全帽款式根據各公司購置不同而有差異，但其安全帽特徵及顏色大致相同，相較於市售機車安全帽，其為了符合消費者喜好而有各式的顏色及花樣，因此機車安全帽之辨識所要訓練與蒐集之資料會更為廣泛。

根據市場分析之案例安全帽辨識特色與差異統整表格 1-3-1，如下：

表 1-3-1 市場案例比較表

案例	特色	劣勢
中國電子警察	以政府道路監視影像取締未戴安全帽之機車用路人	
清波工地安全 AI 影像辨識	具彈性之工地安全帽辨識告警	工地安全帽之辨識侷限，較無法運用於複雜道路

第二章 文獻探討

2.1. You Only Look Once version 8 (YOLOv8)

本組專題目標為處理由多個連續幀所組成之影像，因此在此類即時影像處理的辨識程序中，必須兼顧運行效能與準確率。YOLO(You Only Look Once)為單階段(one stage)的物件偵測演算法，最早由 Joseph Redmon、Santosh Divvala、Ross Girshick 和 Ali Farhadi 於 2016 年共同提出[8]。之後版本的演進在執行效能與準確率上皆有顯著提升，到了 2023 年 Ultralytics 公司已推出第 8 版本[9]。YOLOv8 有較高執行速度與準確率且較直觀的開發介面，符合本組辨識程序需求，因此以 YOLOv8 物件偵測技術應用於機車騎士與乘客和安全帽配戴的檢測，訓練相應的物件偵測模型。

2.1.1. YOLOv8 模型損失函數

YOLOv8 在模型訓練主要使用了三個損失函數來衡量模型預測與真實標記之間的差異，使模型有效學習以提供更準確的預測。其損失函數分別為定位損失函數、分類損失函數和分佈焦點損失函數三者，以下根據損失函數做詳細說明[10]：

(1) 定位損失函數（Bounding Box Regression Loss，簡稱 box_loss）：用以衡量預測的邊界框的準確性，確保預測框與真實框密切匹配。YOLOv8 使用 CIoU loss 實作此函數，而 CIoU loss 除了考慮預測框與真實框的重疊以外，也將其長寬比差異納入比較。

(2) 分類損失函數（Classification Loss，簡稱 cls_loss）：確保模型正確識別每個檢測物體的類別。YOLOv8 以二元交叉熵(Binary Cross Entropy, BCE)損失進行分類處理。

(3) 分佈焦點損失函數（Distribution Focal Loss，簡稱 dfl_loss）：用來解決物件偵測中類別不平衡的問題，也用於邊界框回歸的改善，特別是對邊界模糊或不清晰的物體。dfl_loss 透過估計邊界框坐標的概率分佈，以解決其位置的不確定性。

2.1.2. YOLOv8 模型評估指標

本組欲採用 YOLOv8 模型為處理物件偵測之任務，而物件偵測技術中有多種指標用於評估模型表現，以下為各個性能指標名詞的說明[9]：

- 交並比(Intersection over Union, IoU)：IoU 是一種量化預測邊界框與真實邊界框之間重疊程度的指標，在評估物體定位準確性方面扮演著基本角色。
- 混淆矩陣(Confusion Matrix)：混淆矩陣提供了模型預測與真實樣本之間的細節，顯示每個類別的真陽性(True Positive, TP)、真陰性(True Negative, TN)、假陽性(False Positive, FP)和假陰性(False Negative, FN)的計數。表 2-1-2-1 為混淆矩陣的一種表示形式，如下：

表 2-1-2-1 混淆矩陣

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

- 精確度(Precision)：精確度量化真陽性在所有正向預測中的比例，評估模型避免假陽性的能力。其公式如圖 2-1-2-1 所示，如下：

$$Precision = \frac{TP}{TP + FP}$$

圖 2-1-2-1 精確度公式

- 召回率(Recall)：計算真陽性在所有實際正樣本中的比例，測量模型檢測所有類別實例的能力。其公式如圖 2-1-2-2 所示，如下：

$$Recall = \frac{TP}{TP + FN}$$

圖 2-1-2-2 召回率公式

- F1 Score：F1 Score 是精確度和召回率的調和平均數，提供對模型性能的平衡評估，同時考慮假陽性和假陰性。其公式如圖 2-1-2-3 所示，如下：

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$
$$= 2TP/(2TP + FN + FP)$$

圖 2-1-2-3 F1 Score 公式

- 平均精確度(Average Precision, AP)：AP 計算精確率－召回率曲線下的面積，提供一個單一值來概括模型的精確度和召回率表現。
- 均值平均精確度(mean Average Precision, mAP)：mAP 擴展了 AP 的概念，通過計算多個物體類別的平均 AP 值來進行評估，在多類別物體檢測場景中，有助於提供模型性能的全面評估。
- mAP50：在 IoU 閾值為 0.50 下計算的均值平均精確度，是評估物件偵測準確性的常用指標。
- mAP50-95：在不同的 IoU 閾值下計算的均值平均精確度的平均值，範圍從 0.50 到 0.95（步長為 0.05），提供模型在不同檢測難度水平上較全面的性能評估。

損失函數的變化可用於分析模型訓練的狀況，而評估指標則是模型訓練完衡量模型最終預測結果的表現。本組在訓練機車騎士與乘客和安全帽的物件偵測模型時，可以以上數值為依據進行模型訓練、調適與最終結果評估。

2.1.3. YOLOv8 應用於機車與安全帽辨識

自動化的檢測機車騎士是否佩戴安全帽作為一交通安全的課題，眾多學者運用不同技術研究與解決該問題，而隨著 YOLO 系列演算法的快速發展，其快速、低延遲且保有一定水準準確率的即時影像處理性能，自然成為多位研究人員優先選擇。以下為二則針對機車安全帽道安的相關研究之探討。

Zongxuan Chai 以 YOLOv8 為基礎改良的 YOLOv8-ADown-LSCD，該位學者以 Adown 模組取代原演算法特徵提取過程用到的捲積模組，並將原始偵測頭(detection head)替換成名為 LSCD 的新偵測頭，透過架構的改進從而減少了網路參數量與偵測目標的大小，提升偵測精度與偵測速度，其實驗使用 COCO 資料

集與網路收集的資料共 1721 張圖像作為資料集，並按照 7：1：2 比例分別用於訓練、驗證、測試，訓練之設定 epoch 為 300 對模型進行訓練，消融實驗結果顯示在比較 Precision、Recall、mAP@0.5、mAP@0.5-0.95、FPS 的指標中，YOLOv8-ADown-LSCD 展現最高的性能，證實了其演算法的有效性[11]。

Xiangkui Jiang 和 Libing Pan 基於 YOLOv8n 提出 HD-YOLOv8n 的改善，為了增強模型對環境不規則特徵的適應性並更好地適應目標物件的變形，採用可變形捲積網路 v2(DCNv2)重建原架構骨幹(Backbone)網路中的 Cf2 模組，骨幹網路還整合多維協作注意力(MCA)機制幫助模型更全面、有效地捕捉特徵之間複雜的關係，此外，他們引入了大小為 160x160 小物體偵測層，加強了深層和淺層語意訊息的融合，提高了模型偵測小規模物體的精確度，其實驗使用 Kaggle 公開資料集和多樣角度與情境所拍攝之圖像共 5297 張，依照 8：2 比例分為訓練集和驗證集，以 epoch 為 110、初始學習率為 0.001 進行訓練，消融實驗結果比較 Precision、Recall、mAP@0.5、mAP@0.5-0.95 等評估數值，HD-YOLOv8n 皆有所提升[12]。

二則研究都以 YOLOv8 進行模型改進以符合對應的應用情境，並使用 Precision、Recall、mAP50、mAP50-95 等物件偵測模型評估指標進行比較，最終得出較佳的模型。本組於機車騎士與乘客和安全帽之模型訓練上，也應善用相關的模型訓練方法，找出較適合的模型以滿足辨識流程之需要。

2.2. ONNX 與 ONNX Runtime

2.2.1. Open Neural Network Exchange (ONNX)

開放神經網路交換（Open Neural Network Exchange，以下簡稱 ONNX）是機器學習開放式文件格式，用於儲存訓練好的模型，ONNX 定義了一組通用的機器學習與深度學習基本運算子以及通用的檔案格式，讓 AI 開發人員能在多種框架、工具、運行時和編譯之間使用模型(如圖 2-2-1-1)，ONNX 由 Microsoft、Facebook、Amazon、IBM 等公司所組成之社群共同開發與維護，其原始碼提供於 GitHub，並支援多種框架的運用及轉換，如：Caffe2、PyTorch、TensorFlow、Keras、SciKit

Learn[13]……

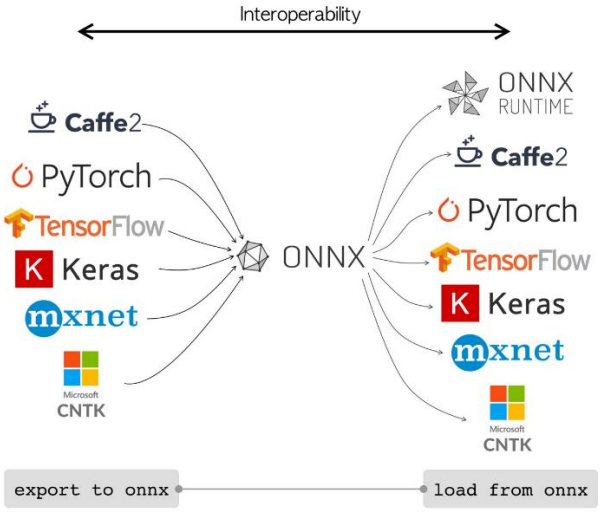

圖 2-2-1-1 ONNX 模型匯出與轉換[14]

2.2.2. ONNX Runtime

ONNX Runtime 是一個跨平臺機器學習模型加速器，用於推理和訓練模型可優化運行效能，只需將模型匯出或轉換為 ONNX 格式，透過 ONNX Runtime 載入和運行即可有與原始框架相比更高的推理效能改進，ONNX Runtime 除了可在不同硬體和作業系統最佳化的運行（如圖 2-2-2-1 所示），也提供多種程式語言的 API，如：C++、Python、Java 等，於模型部署上有更多的彈性[15]。

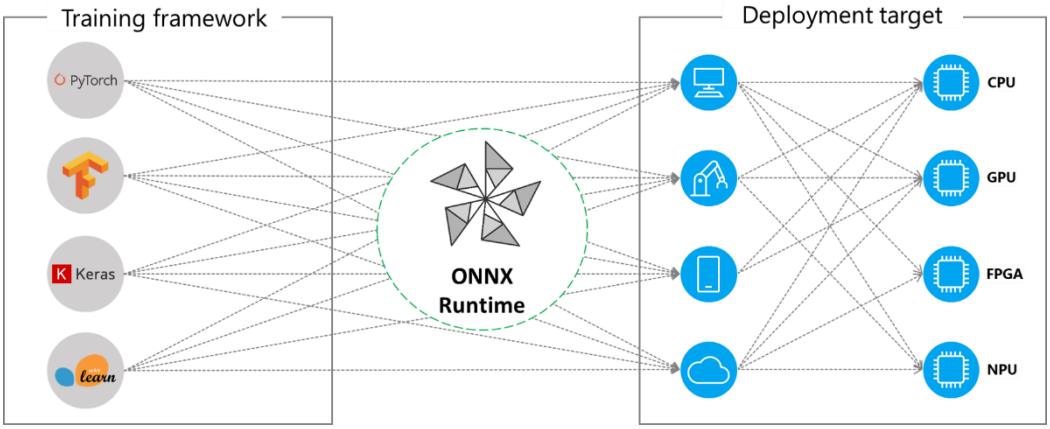

圖 2-2-2-1 ONNX Runtime 不限模型框架與跨平臺特性[15]

ONNX Runtime 大量運用於 Microsoft 的服務，如：Bing、Office 和 Azure AI 等，其服務報告指出 CPU 的效能平均有 2 倍提升，除了 Azure Machine Learning

服務之外，ONNX Runtime 也能在支援機器學習工作負載的其他產品中執行，如：Windows 中的 Windows Machine Learning、Azure SQL 產品的 Azure SQL Edge 和 Azure SQL Managed Instance 以及 ML.NET 等[16]。

在現有的研究報告中，Murat Sever 和 Sevda Öğüt 兩位學者以平均功率計算模型來比較機器學習推理中各種框架的延遲，該模型架構使用 Python 的 PyTorch 之 2 層神經網路創建，實驗實作步驟如圖 2-2-2-2 所示，模型被轉換為 Torch Script 和 ONNX 格式，並以 C++程式進行速度比較，Torch Script 在 CPU 和 GPU 上使用 LibTorch 運行，而 ONNX 除了在 CPU 和 GPU 使用 ONNX Runtime 外，另外於 GPU 上以 TensorRT 運行[17]。

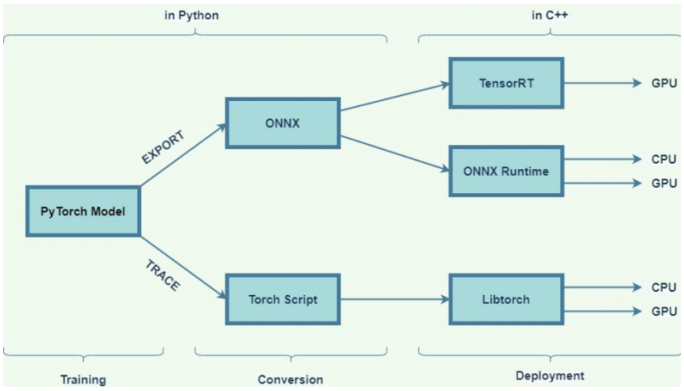

圖 2-2-2-2 實作過程[17]

其實驗結果比較如圖 2-2-2-3，顯示 ONNX 格式搭配 TensorRT 的推理速度最快，而儘管 ONNX 在 ONNX Runtime 上只使用 CPU 運作，但推理速度也有一定水準[17]。

Framework	Inference Execution Time (ms)
Torch Script + Libtorch (CPU)	1.86176
Torch Script + Libtorch (GPU)	1.88932
ONNX + ONNX Runtime (CPU)	1.18248
ONNX + ONNX Runtime (GPU)	2.90259
ONNX + TensorRT (GPU)	0.39293

圖 2-2-2-3 運行時間比較[17]

綜合上述應用實例與實驗比較，將模型轉換為 ONNX 格式並於 CPU 運行 ONNX Runtime，可加快模型推理速度，而本組影像辨識可運用此方式配合 YOLOv8 模型進行圖像物件偵測，以取得較高的辨識效能。

2.3. 光學字元辨識(OCR)與車牌辨識

2.3.1. 光學字元辨識(OCR)

光學字元辨識（Optical Character Recognition，以下簡稱 OCR）是將影像轉換為機器可讀格式的技術，透過自動化的識別把影像資料轉換為文本，減少人工輸入錯誤，也提高了資料處理效率。以下為具體的 OCR 運作程序[18]：

(1) 影像獲取：運用掃描器將實體文件轉換為二進位的影像資料，接著即進行影像二值化(Thresholding)，也就是透過 OCR 軟體分析該資料並將淺色區域視為背景，深色區域歸為文字。

(2) 影像預處理：清理影像和修正錯誤，增強影像品質以便後續的文字辨識，其中包含了傾斜修正、去斑點、平滑化、非必要方框及線條清除等程序。

(3) 文字辨識：分為模式比對與特徵擷取等演算法。模式比對以字元為單位分離出個別的字符影像，與相似之已儲存字符模式比對，此方式僅在字模與字符影像皆屬類似的字型與比例才有效用，較適用於已知字型格式之文件影像掃描。特徵擷取是將字符細分或拆解為特徵，如：線條、閉環、線條方向、線條交叉點等，利用特徵在已儲存字符中找出最符合或相近的字元。

(4) 後處理：分析完後，系統將擷取的文字資料轉換為電腦化的檔案。其中也包含了糾正和修改文字，藉由拼字檢查或考慮上下文等方式，以更正分析出的文本。

根據 AWS 對 OCR 的解釋，OCR 的類型可分為以下四點[18]：

- 簡單 OCR 軟體：將多種不同字型的影像視為模板並以資料庫或檔案等形式儲存，使用模式比對的演算法，將文字影像逐字元與儲存資料比對。若辨識目標為不限字型或手寫樣式，因為無法對每個可能類型擷取與儲存，因此具有侷限性。
- 智慧字元辨識(Intelligent Character Recognition, ICR)軟體：藉由機器學習軟體，訓練機器展現像人類一樣行為來辨識圖像，透過神經網路在多個層級上

分析文字，並重複處理影像，以尋找不同影像屬性，如：曲線、直線、交叉點和閉環等，最後結合所有層級的分析獲得最終結果，此類辨識通常一次處理一個字元影像。

- 智慧文字辨識(Intelligent Word Recognition)：運作方式與 ICR 相同，但處理的是整個文字影像，不是一次處理一個字元影像。
- 光學標記辨識(Optical Mark Recognition)：針對文件中的標誌、浮水印和其他符號進行辨識，應用於考試答案卡、問卷等讀取作業。

2.3.2. OCR 應用於車牌辨識

OpenALPR 是一個以 C++編寫的開源自動車牌辨識函式庫，於 GitHub 可查看開放原始碼，該函式庫提供 C++、C#、Java、Node.js、Go 和 Python 等程式語言連接 API，使用 OpenCV 和 Tesseract OCR 函式庫識別影像中的車牌，其詳細處理程序如下[19]：

(1) 偵測(Detection)：首先從影像中尋找潛在的車牌區域，使用 Local Binary Patterns(LBP)演算法與 cv2::CascadeClassifier 類別實作，輸出所有可能車牌的座標與長寬，並將這些區域影像交由後續程序進行辨識。

(2) 二值化(Binarization)：接收到可能的車牌區域後，運用 Wolf-Jolien 及 Sauvola 方法，應用不同參數，將影像轉為黑白。

(3) 字元分析(Character Analytics)：在二值化後的影像尋找字符，以連通且長寬較為相近的斑塊為目標來查找，若無任何字符即視為該區域無車牌號碼。

(4) 車牌邊緣(Plate Edges)尋找：以霍夫轉換(Hough Transform)更為精確的找出車牌上下左右之邊緣。

(5) 傾斜校正(Deskew)：透過車牌邊緣資料將車牌影像修正為標準尺寸和方向。

(6) 字元分割(Character Segmentation)：利用垂直值方圖尋找字元的間隙以分離出車牌中所有字元，此階段也排除了字元框中較小、不連續的斑點和高度不足的字符區域。

(7) OCR：以 Tesseract OCR 獨立分析每個字元，對於每個字元影像，計算所有可能的字元與置信度。

(8) 後處理(Post Processing)：輸出所有可能的 OCR 字元和置信度列表，確定最佳的車牌號碼組合，並將低於特定閾值的字元取代為空白字元。後處理也可針對地區車牌規則進行驗證，以得出符合標準之最佳結果。

針對臺灣車牌辨識處理，張逸忠與李美億於《影像辨識實務應用：使用 C#》一書中，利用 C#實作 OCR 車牌辨識，以下為處理程序的概述[20]：

(1) 簡化影像：此階段是將影像資訊簡化以聚焦於車牌目標。首先，在 RGB 三原色中選出一種對辨識預期目標最有效的灰階亮度，而自然光中整體代表性最高者為綠光，因此只保留綠光資料，將影像資料灰階化，從 RGB 三維陣列轉換為一維。接著將灰階影像轉換為黑白圖，透過影像二值化可將原本獨立塊狀之車牌字元分割出來，使得字元呈現黑色，而背景變為白色。之後藉由二值化影像畫出輪廓線，驗證黑色塊狀目標群集之大小、形狀是否像字元目標，找出並鎖定車牌目標字元。

(2) 正規化目標影像：由於拍攝角度的不同，目標字元會有一定的傾斜與變形，且其大小與標準字模不一致，因此須正規化影像，以幾何計算把目標從任一四邊形投射為矩形，並縮放字元使其符合標準字元模型之大小。

(3) 字模比對與資料確認：完成正規化後即可用事先建立之個別字元模型比對出車牌內容，而比對結果若與真實車牌號碼規則不相符，則要修正內容，確保最終輸出之車牌的合理性。

根據本組影像辨識目標，在檢測出未戴安全帽的騎士或乘客後，須識別違規者車輛之車牌號碼。上述二者 OCR 實作皆可滿足車牌辨識之需要，但前者用到 LBP 演算法與 Tesseract OCR 分別實作車牌物件偵測與字元辨識，而後者僅針對影像資料進行幾何計算與比對，所以後者的辨識速度會比前者快，因此本組使用後者之處理程序，實現車牌號碼的檢測。

第三章 研究方法

3.1. 研究流程

圖 3-1-1 為本組研究流程，首先是系統架構規劃，到辨識流程的規劃後，即開始系統開發程序，系統開發過程分別針對網頁界面、資料庫設計以及系統辨識流程實作等方面進行，辨識流程實作過程中首先訓練出 YOLOv8 模型，接著開發 OCR 車牌辨識，之後以 C++程式實作辨識流程，最終，將所有元件整合，並完成系統成果。

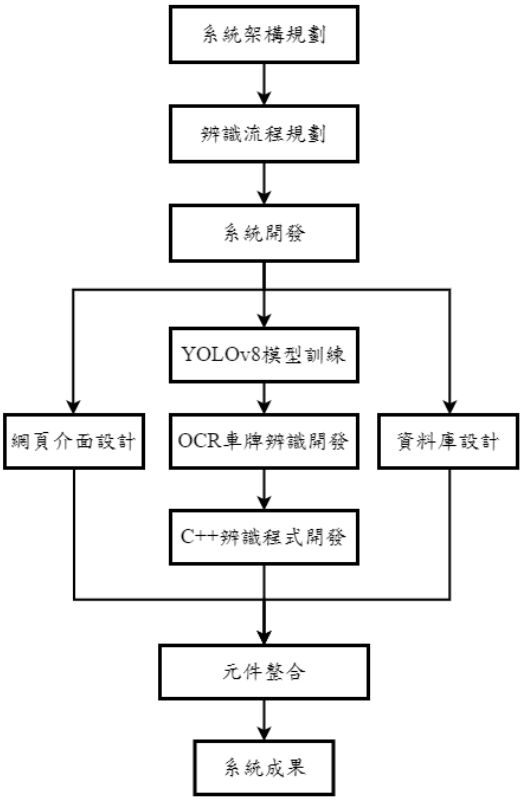

圖 3-1-1 研究流程

圖 3-1-1 中的 YOLOv8 模型訓練流程如圖 3-1-2 所示，首先收集圖片，再依照 YOLOv8 資料格式對圖片標註，之後根據採用的模型驗證方法拆分資料為訓練集、驗證集和測試集，而後訓練模型，並依據模型評估結果調整超參數重新訓練以找出較佳模型結果，最後，將模型格式從 PyTorch 框架轉換為 ONNX 格式，完成模型訓練。

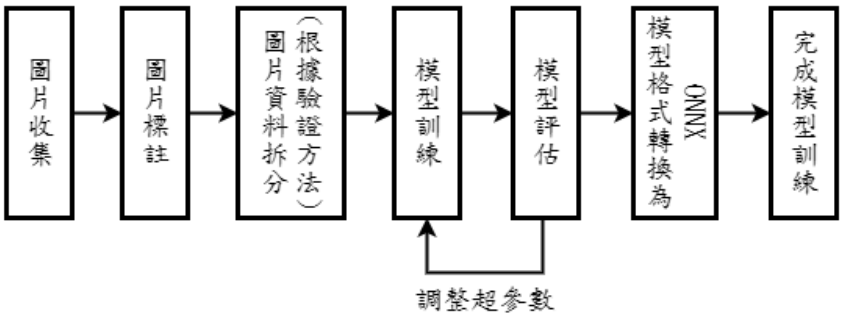

圖 3-1-2 YOLOv8 模型訓練流程

3.2. 系統架構與技術

本組規劃之系統是基於網頁（Web-based）應用服務，由 Apache、PHP 模組、C++執行檔、MariaDB 等元件所構成，系統架構圖如圖 3-2-1 所示。使用者透過網頁上傳欲辨識的圖片或影片至 Apache，接著由 PHP 將來源檔案儲存於伺服器後，將其資料存入 MariaDB，並呼叫 C++程式執行，將結果網頁雛型回傳。而 C++將辨識資料存入 MariaDB，並開始辨識，期間 C++會不斷儲存結果和更新辨識進度，同時使用者端透過 AJAX 技術定時檢查辨識狀態及結果，由 PHP 自 MariaDB 查詢後並回傳資料，最終呈現給使用者。

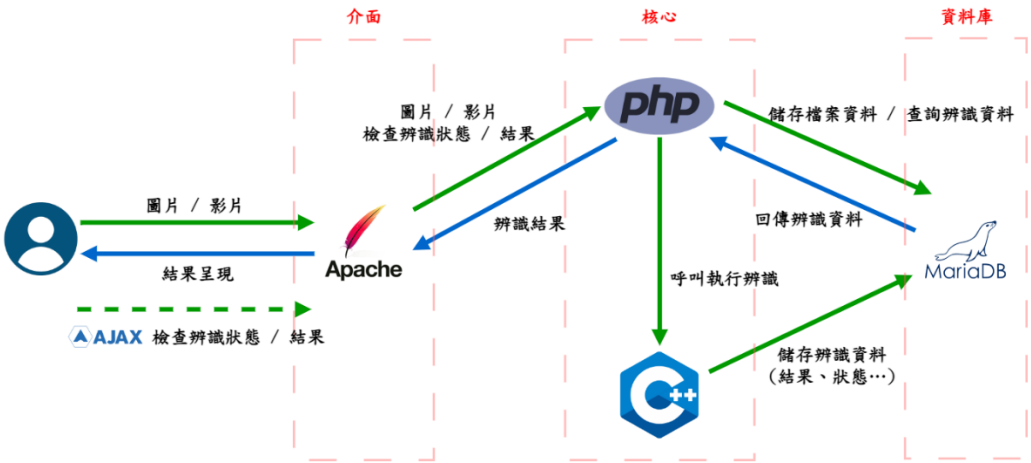

圖 3-2-1 系統架構圖

圖 3-2-2 系統技術

系統所運用到的技術如圖 3-2-2 所示，以下根據介面、核心（應用）、資料庫等層面做描述：

- 介面(Interface)：HTML、CSS、JavaScript 為網頁基本要素，以 HTML 定義網頁架構，CSS 設計外觀、JavaScript 處理使用者於網頁的操作行為。網頁另外引入 jQuery 函式庫，以簡化 JavaScript 的撰寫。為了使網頁的使用更加平順、流暢，採用非同步 JavaScript 和 XML（AJAX）的技術，讓網頁不需重新整理就可以 JavaScript 與伺服器交換資料、變更網頁的內容，達到更好的使用體驗。
- 核心(Application)：PHP 程式主要負責處理使用者所上傳的檔案與回覆 AJAX 的要求，在其間與資料庫建立連線以取得所需資料，最終產生網頁或 JSON 資料。C++、OpenCV、YOLOv8、ONNX Runtime 即為未戴安全帽之辨識技術，而 Python 則主要運用於 YOLOv8 模型訓練。
- 資料庫(Database)：採用 MariaDB 關聯式資料庫，儲存檔案上傳相關資料，包含檔案詳細資訊、當前辨識的狀態等……

3.3. 系統辨識流程

本組系統辨識流程使用 C++程式語言進行處理，首先透過 OpenCV 函式庫依序讀取影像幀資料，經由首個 YOLOv8 物件偵測模型推理出機車騎士與乘客

的目標範圍，再將其資料交由第二個 YOLOv8 物件偵測模型推理安全帽的佩戴與否。為了部署模型與加快推理運行速度，上述兩者物件偵測的辨識皆以 ONNX 格式之 YOLOv8 模型，使用 ONNX Runtime 函式庫在 CPU 上進行模型推理。經第二個物件偵測判斷出未戴安全帽後，會對騎士影像資料以 OCR 車牌辨識擷取出車牌號碼，再將該幀以圖片形式存入伺服器，而其相關資料包括：檔案位置、影格編號、影格的時間點和車牌等皆會儲存於資料庫。期間辨識處理後的影像畫面，也會存入影像檔案，輸出辨識後的影片檔案以供系統介面之結果呈現。如此對所有幀處理完後，方能完成系統辨識流程，並達到本組辨識需求與目的。

以上辨識處理流程展示於圖 3-3-1：如下：

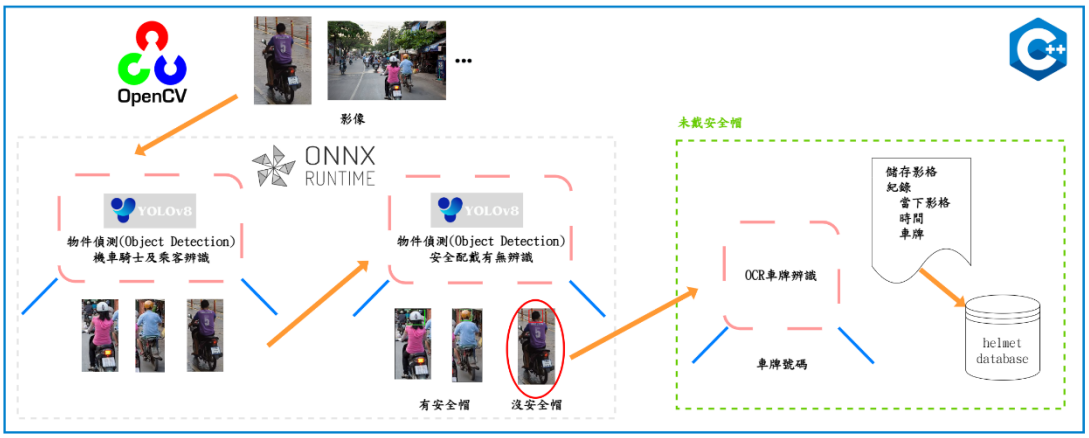

圖 3-3-1 系統辨識流程圖

3.4. YOLOv8 模型訓練

本文於 3.3 節提及系統辨識流程，其中使用了兩個 YOLOv8 模型分別針對機車騎士及乘客與安全帽配戴有無進行物件偵測的任務，而為獲取符合需求且準確率較佳的模型，本組依圖片收集、圖片標註……到最後模型訓練、模型格式轉換等步驟（圖 3-1-2），訓練出 YOLOv8 物件偵測模型。此節為模型訓練過程之詳細說明，將以開發環境、資料收集、資料集拆分、模型訓練參數、最終模型評估、模型格式轉換等方面進行描述。

3.4.1. Python 開發環境

本組研究初期使用 Google Colaboratory 雲端環境進行程式測試，而後在本機端建置相關環境並開發系統，以下是本機端之 Python 開發環境的說明。

本機端以 GPU 硬體進行運算可加快模型訓練的速度，要使用 Nvidia GPU 運行前須處理好 CUDA 的設置，表 3-4-1-1 為 GPU 規格與相關工具版本：

表 3-4-1-1 GPU 與其環境設定

NVIDIA GeForce RTX 3070 Laptop GPU	
Driver	546.80
CUDA	12.3
CUDA Toolkit	11.6, V11.6.55

本組欲訓練 YOLOv8 模型，需要 Python 開發環境。Conda 有方便的套件與環境管理功能，使開發者更便利地進行程式測試與開發。Miniconda 是一免費的 Anaconda 發行版之迷你安裝版本，僅包含 Conda、Python、它們所依賴的套件，以及少量其他有用的套件[21]。因此，本組選用 Miniconda 以建置 Python 開發環境，利用 Conda 創建虛擬環境並安裝所需套件，其中 PyTorch 套件之版本須相容於表 3-4-1-1 中 CUDA 版本。所用作業系統、軟體、虛擬環境與主要 Python 套件版本整理如表 3-4-1-2：

表 3-4-1-2 開發環境與主要 Python 套件版本

作業系統	
Windows	x64, 11
軟體	
Miniconda3	py312_24.5.0-0
虛擬環境	
Python	3.8.19
主要套件	
pytorch	1.12.1

表 3-4-1-2 開發環境與主要 Python 套件版本（續）

主要套件	
torchvision	0.13.1
torchaudio	0.12.1
ultralytics	8.2.71
numpy	1.24.4
labelImg	1.8.6
scikit-learn	1.5.2
onnx	1.16.2

3.4.2. 資料集

3.4.2.1. 機車騎士與乘客資料集

機車騎士與乘客的圖像本組從 Roboflow 與 Kaggle 公開資料集中收集圖片，總共收集 2777 張含有機車騎士與乘客的圖片，其中本組以水平翻轉的方式對 npk7264 提供之 641 張圖做資料增強(Data Argumentation)，將其資料量擴大為二倍。表 3-4-2-1-1 為資料集之詳細資訊：

表 3-4-2-1-1 機車騎士與乘客資料集來源

來源	提供者	參考文獻索引	數量(張)
Kaggle	Abhishek Singh Yadav	[22]	341
	npk7264	[23]	863
Roboflow	Helmet Detection	[24]	1282 （含水平翻轉）
	Object Detection HelmetsLicense	[25]	291
合計			2777

3.4.2.2. 安全帽資料集

安全帽的圖像本組除了從 Roboflow 與 Kaggle 公開資料集中收集圖片，也另外使用自行拍攝的影片之幀圖像，總共收集 10500 張含有有戴安全帽與未戴安全帽之案例。表 3-4-2-2-1 為資料集之詳細資訊：

表 3-4-2-2-1 安全帽資料集來源

來源	提供者	參考文獻索引	數量(張)
Kaggle	Abhishek Singh Yadav	[22]	343
	npk7264	[23]	875
	Moazzim Ali Bhatti	[26]	242
Roboflow	Helmet Detection	[24]	635
	Object Detection HelmetsLicense	[25]	293
	CSGO Head Detection	[27]	1630
自行拍攝			6482
合計			10500

3.4.2.3. 圖片標註

收集完圖片後，要對對應資料集標註欲訓練之目標位置，本組使用 labelImg 工具進行圖片標註，以輸出符合 YOLO 格式之標註檔案。在機車騎士與乘客資料集中，總共有 1 個類別為 scooter，表示機車騎士與乘客；安全帽資料集中，總共有 2 個類別分別為 helmet 及 no_helmet，代表有戴安全帽與未戴安全帽之人的頭部。於 labelImg 標註的範例如圖 3-4-2-3-1 和圖 3-4-2-3-2。

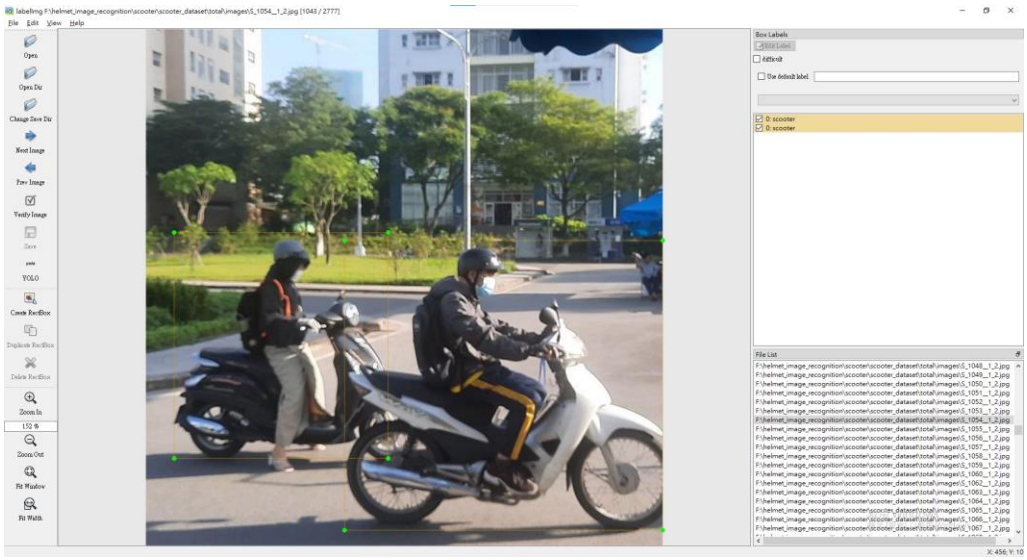

圖 3-4-2-3-1 機車騎士與乘客目標之標註

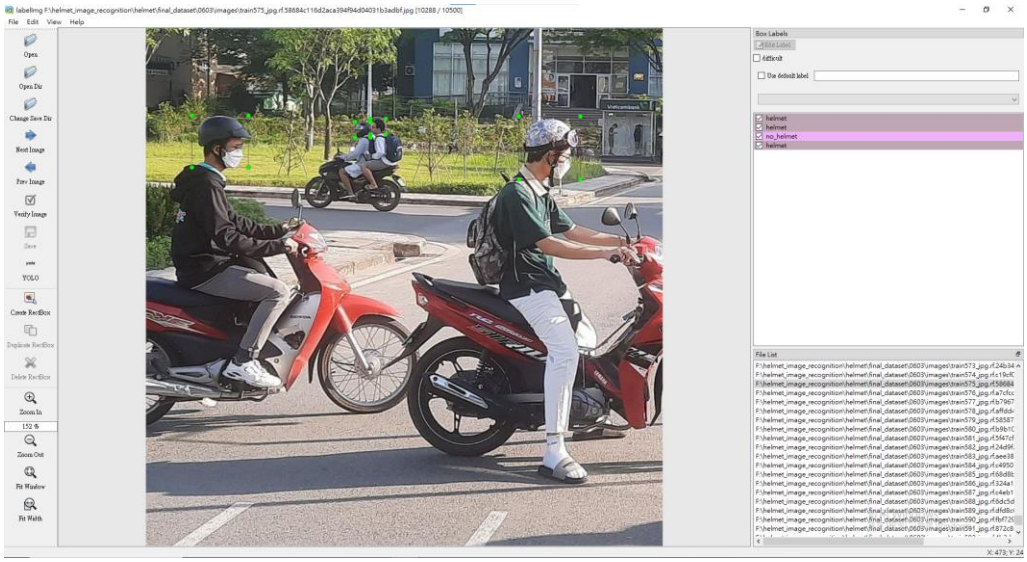

圖 3-4-2-3-2 安全帽目標之標註

3.4.3. 資料集拆分

3.4.3.1. 機車騎士與乘客資料集拆分

對機車騎士與乘客資料集，本組使用 Hold-out Validation method 拆分資料集與評估模型[28]。訓練、驗證、測試分別以 0.8、0.1、0.1 的比例切割總數為 2777 的圖片資料，拆分後訓練集為 2221 張、驗證集為 277 張、測試集為 279 張圖片。資料集之拆分概況整理如表 3-4-3-1-1：

表 3-4-3-1-1 機車騎士與乘客資料集劃分

機車騎士與乘客資料集（2777 張）		
Hold-out Validation method		
train	0.8	2221
test	0.1	277
validation	0.1	279

3.4.3.2. 安全帽資料集拆分

對安全帽資料集，本組使用 K-Fold Cross-Validation method 拆分資料集與評估模型[28]。本組以 sklearn 套件裡 KFold 函式將資料集拆分，設 K 為 10(n_split)，並以 random_state 為 0，最終分出 10 折資料集(split_1~split_10)，每一折(Fold)資料集中訓練和驗證資料以 0.9、0.1 的比例切割，因此，訓練集的圖片數有 9450 張，驗證集則為 1050 張。安全帽資料拆分示意圖如圖 3-4-3-2-1：

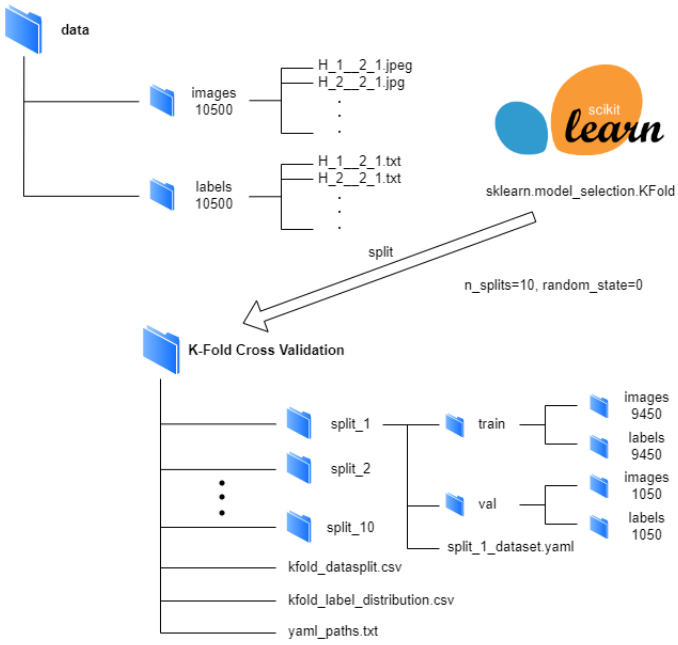

圖 3-4-3-2-1 安全帽資料集劃分

3.4.4. 模型訓練參數

YOLOv8 提供多種不同的物件偵測模型，模型參數量由小至大依序為

YOLOv8n、YOLOv8s、YOLOv8m、YOLOv8l、YOLOv8x，在 COCO 資料集中的表現，參數量越小者推理速度較快，但準確率較低，反之，參數量越大推理速度較慢，但準確率較高，上述模型相關效能評估如表 3-4-4-1 所示，所有模型皆以 640x640 像素之圖片進行評估[14]。為了以較快的速度運行系統辨識，本組選擇 YOLOv8n 物件偵測模型進行訓練。

表 3-4-4-1 YOLOv8 模型於 COCO 資料集效能評估[14]

Model	**mAP50-95**	**Speed CPU ONNX (ms)**	**Speed TensorRT(ms)**	**params**	**FLOPs(B)**
YOLOv8n	37.3	80.4	0.99	3.2	8.7
YOLOv8s	44.9	128.4	1.2	11.2	28.6
YOLOv8m	50.2	234.7	1.83	25.9	78.9
YOLOv8l	52.9	375.2	2.39	43.7	165.2
YOLOv8x	53.9	479.1	3.53	68.2	257.8

機車騎士與乘客物件偵測模型之訓練，本組選用 YOLOv8n 預訓練模型(pretrained model)，並將 epoch 設為 100、image size 定為 640、batch size 為 16、workers 為 4，其餘參數用 YOLOv8 預設之設定值，以 GPU 設備進行訓練。

安全帽物件偵測模型之訓練，本組選用 YOLOv8n 預訓練模型(pretrained model)，並將 epoch 設為 150、image size 定為 640、batch size 為 32、workers 為 4，其餘參數用 YOLOv8 預設之設定值，對 10 折 K-Fold Cross-Validation method 拆分後的資料集以 GPU 進行訓練。

以下二張截圖為機車騎士與乘客和安全帽模型訓練 Python 程式碼，分別為圖 3-4-4-1 與圖 3-4-4-2，而圖 3-4-4-2 為第 5 折(split_5)經過 K-Fold 拆分之資料集訓練程式。

```
from ultralytics import YOLO

device = 0

def new train():

        # Load a model
        model = YOLO('yolov8n.pt')

        # Train the model
        results = model.train(data='C:/Users/NUTC/Desktop/helmet/scooter dataset/scooter.yaml', epochs=100, imgsz=640, device=device, batch=16, workers=4,
project="C:/Users/NUTC/Desktop/helmet/results/", name="scooter")
```

圖 3-4-4-1 機車騎士與乘客訓練程式

```
from ultralytics import YOLO

split = "split 5"
device = 0

def new train():

        # Load a model
        model = YOLO('yolov8n.pt')

        # Train the model
        results = model.train(data=f'C:/Users/NUTC/Desktop/helmet/{split}/{split} dataset.yaml', epochs=150, imgsz=640, device=device, batch=32, workers=4,
project="C:/Users/NUTC/Desktop/helmet/results/", name=f"yolo train result {split}")
```

圖 3-4-4-2 安全帽訓練程式(split_5)

3.4.5. 模型評估

本節將以 Precision、Recall、F1 Score、mAP50、mAP50-95 等指標為基準，對機車騎士與乘客和安全帽模型之表現進行評估。

3.4.5.1. 機車騎士與乘客模型評估

機車騎士與乘客模型之訓練訓練結果，如表 3-4-5-1-1 所示，Precision、Recall、F1 Score、mAP50、mAP50-95 分別達到 0.967、0.983、0.97、0.991、0.905。

表 3-4-5-1-1 機車騎士與乘客模型評估

Metrics	Precision	Recall	F1 Score (with conf)	mAP50	mAP50-95
	0.967	0.983	0.97(0.297)	0.991	0.905

3.4.5.2. 安全帽模型評估

所有 K-Fold 拆分後的安全帽資料集之訓練結果，以表 3-4-5-2-1 陳列各折指標數值，如下：

表 3-4-5-2-1 安全帽模型評估（共 10 折）

Metrics	n_split									
	1	2	3	4	5	6	7	8	9	10
Precision	0.929	0.921	0.926	0.940	0.907	0.926	0.918	0.925	0.930	0.918
Recall	0.857	0.854	0.859	0.881	0.860	0.862	0.864	0.867	0.868	0.854
F1 Score	0.89	0.89	0.89	0.91	0.88	0.89	0.89	0.90	0.90	0.88
Max F1 Score with conf	0.396	0.351	0.382	0.405	0.339	0.341	0.378	0.368	0.393	0.368
mAP50	0.925	0.918	0.925	0.94	0.921	0.928	0.924	0.928	0.931	0.924
mAP50-95	0.614	0.616	0.622	0.637	0.611	0.628	0.614	0.620	0.627	0.613

根據 K-Fold Cross-Validation method，最終評估數值為表 3-4-5-2-1 中各指標之平均值，因此，Precision、Recall、F1 Score、mAP50、mAP50-95 平均後分別達到 0.924、0.863、0.89、0.926、0.620，如表 3-4-5-2-2 所示：

表 3-4-5-2-2 安全帽模型評估（平均）

Average	Precision	Recall	F1 Score	mAP50	mAP50-95
Metrics	0.924	0.863	0.89	0.926	0.620

3.4.6. 格式轉換

根據 3.3 節系統辨識流程，為了讓訓練出的模型於 ONNX Runtime 運行，須將所有模型權重由 PyTorch 格式（副檔名為.pt）轉換為 ONNX 格式（副檔名為.onnx）。轉換程式範例截圖如圖 3-4-6-1 所示，把訓練最佳權重檔案(best.pt)轉換為 best.onnx，並設定允許推理時輸入動態大小的圖片。

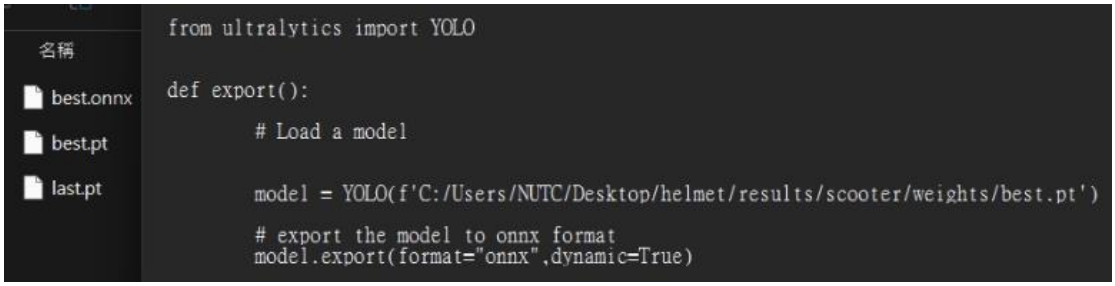

圖 3-4-6-1 轉換程式（機車騎士與乘客模型）

綜合 YOLOv8 物件偵測模型訓練過程，整理如圖 3-4-6-2 所示：

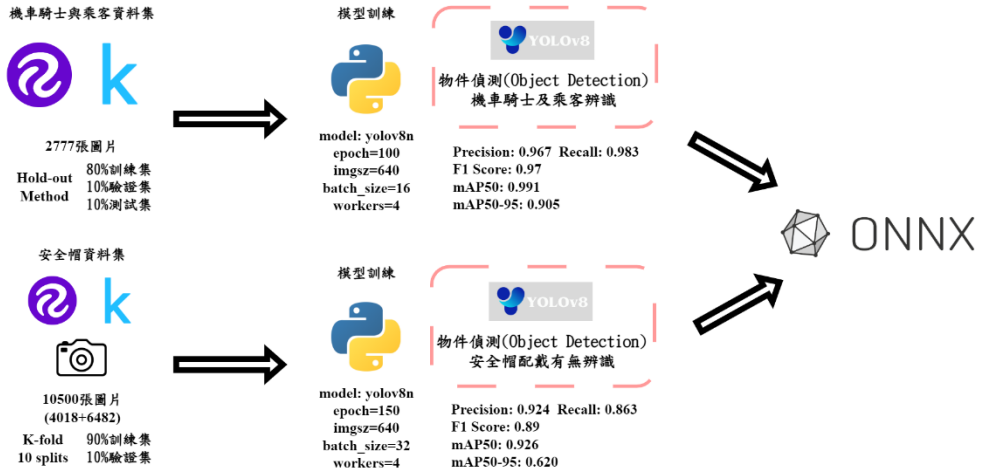

圖 3-4-6-2 YOLOv8 模型訓練概觀圖

3.5. OCR 車牌辨識

本組以《影像辨識實務應用：使用 C#》一書為範本，實作 OCR 車牌辨識之程式[20]。本組設計之程式流程如圖 3-5-1，如下：

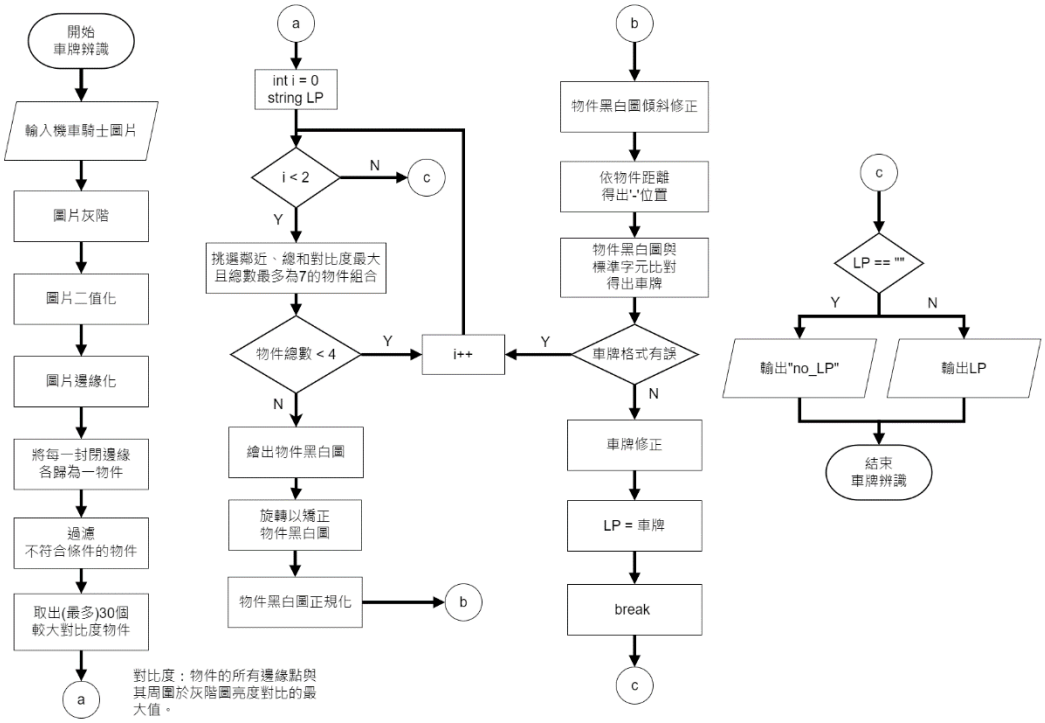

圖 3-5-1 OCR 車牌辨識流程圖

以下依據 OCR 車牌辨識流程做詳細描述：

(1) 機車騎士圖片輸入：系統辨識出未戴安全帽之行為人，則須將騎士圖片資料進行辨識以得出其車牌號碼。

(2) 灰階：將輸入之圖片以綠光做灰階化，由 RGB 陣列轉為一維陣列資料。

(3) 二值化：將圖片以 40x40 區塊切割，並計算每一區塊中像素總和之平均值做為二值化門檻值（閾值），若像素超過閾值使其變為白色(255)，反之則視為黑色(0)，如此對所有區塊處理。

(4) 邊緣化：為了對所有在視覺上獨立的黑色區塊，提取其資訊包裝為物件形式，使程式得以對其進行後續辨識處理，而有邊緣化與建立物件的程序。首先，根據二值化的圖像檢查所有黑色像素，若其周圍存在白色像素，則保持該像素為黑色，否則變為白色，如此可留下連通的黑色區塊之輪廓。

(5) 建立物件：以邊緣圖為基礎，使用氾濫式演算法(Flood Fill Algorithm)提取封閉（連通）之邊緣像素，並將其建立為一物件(Object)，該物件包含了該黑色區塊之高度、寬度、座標範圍、中心點、所含之邊緣像素的座標、與周圍像

素比較之對比度等資料。

(6) 物件條件過濾：確認物件高度與寬度符合條件，高度須介於 10 至 80px；寬度須介於 2 到 80px，不符則剔除該物件。

(7) 取出對比度較大之物件：依據對比度排序物件，取出最多 30 個對比度較大者。對比度是由物件的所有邊緣點與其周圍於灰階圖亮度對比的最大值。

(8) 挑選物件組合：選定一物件為基準，搜索其中心鄰近範圍內（±2.5 倍基準物件之寬度、±1.25 倍基準物件之高度）對比度較大的物件（包含自身最多為 7 個），且被選中之物件高度須落在基準高度的 0.55 到 1.5 倍，其寬度須小於等於基準高度，如此對所有物件確認其鄰近組合，最終選出組合中總和之對比度最大者。若該組合的物件數小於 4，則重複第 8 步驟。若 2 次執行步驟 8 至 13 未得出車牌，則視為無法辨識或無車牌。

(9) 物件黑白圖繪製與旋轉：透過二值化圖像與物件邊緣資料，繪製每一物件之黑白圖。再透過最左與最右方物件中心座標之差計算傾斜角度，以順時針或逆時針的方向旋轉所有黑白圖。

(10) 物件黑白圖正規化：正規化前要先計算第 2 大與第 3 大之物件高度與寬度之平均，以其為基準比例將所有黑白圖縮放為 25x50 的大小。

(11) 黑白圖傾斜修正、得出「-」位置：透過上下平移錯位黑白圖，修正影像因立體空間造成的傾斜。計算相鄰物件中心的直線距離，若為最大值則得知車牌之「-」字元位於該處。

(12) 字模比對，得出車牌：物件黑白圖與事先建立的字模比對，計算黑色像素符合比例並得出符合度，最高者即是該物件最匹配之字元。字模由總共有 74 個標準字元，其中有 2 種分為 6 位元與 7 位元的車牌（0 至 9、A 到 Z），另外有 2 個特殊的 6 與 9 字元。

(13) 車牌格式檢查：確認「-」位置，也檢查總合之符合度是否大於等於 600 與字元總數（含「-」）是否大於等於 5，若不符合則回到第 8 步驟。

(14) 車牌修正：臺灣車牌格式分為 5 位元、6 位元與 7 位元車牌，其中以「-」為

分界而有以下幾種代碼與代號的組合：2-3、3-2、2-4、4-2、3-3、3-4 等。以「-」將車牌分為左半部及右半部，其中較長的區段一定都是數字。7 位元車牌前 3 字元一定是英文，後 4 字元為數字。採用以上格式檢查，將確定是數字或英文區段的字元修正。如：8 換成 B、2 換成 Z、O 換成 0、I 換成 1……

(15) 輸出車牌：經由上述步驟若未得出車牌號碼則以「no_LP」輸出，否則輸出所辨識之車牌號碼。

綜合車牌辨識流程說明，以圖 3-5-2 展示車牌辨識圖像處理的變化過程示意，如下：

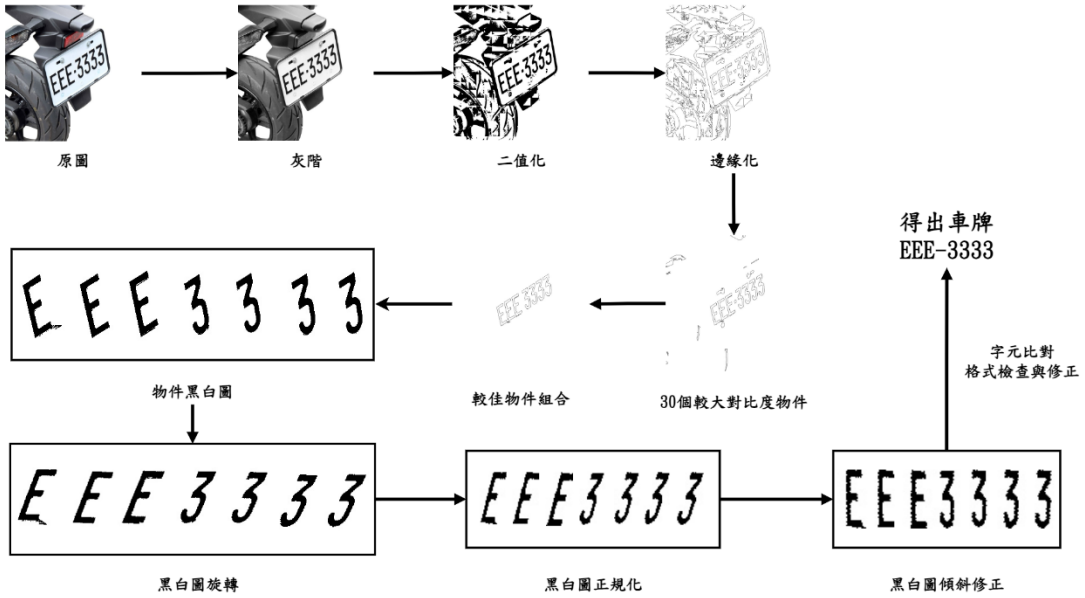

圖 3-5-2 OCR 車牌辨識示意圖

3.6. C++辨識程式

本組 C++辨識程式開發是以 Microsoft Visual Studio Community 2019 進行程式撰寫和執行檔編譯，程式中引入了 OpenCV、ONNX Runtime、Mysql-Connector-C 等函式庫，其中 OpenCV 與 ONNX Runtime 用於影像處理，而 MySQL-Connector-C 則是處理 C++與資料庫的溝通。函式庫的版本對應如表 3-6-1，如下：

表 3-6-1 C++函式庫對應表

函式庫	
OpenCV	4.10.0
ONNX Runtime	x64-1.16.3
Mysql-Connector-C	6.1.11-winx64

此外，C++執行檔須在 Windows x64 之作業系統運行，使所有函式庫之功能得以正常運作。而辨識執行檔在執行辨識時，以 PHP 的 popen 和 pclose 函式建立和關閉行程，以執行運行 C++的系統指令。

第四章 系統成果

4.1. 系統使用環境

對應 3.2 節的系統架構，本組系統所用元件包含 Apache 網頁伺服器、PHP 模組、MariaDB 資料庫管理系統、C++辨識程式，其中 C++辨識程式於 3.6 節提及，須以 Windows x64 的作業系統執行該程式執行檔。因此本組開發之系統建置於 64 位元 Windows 10 作業系統，並使用 Xampp 設置 Apache 網頁伺服器、PHP 模組、MariaDB 資料庫管理系統。各系統元件的版本對應表詳見表 4-1-1：

表 4-1-1 系統元件的版本對應表

系統元件	版本及類型
Microsoft Windows 作業系統	10，x64
Xampp	8.0.28
Apache	2.4.56
PHP	8.0.28
MariaDB	10.4.28

系統以 Apache 伺服器運行 PHP 模組，使用者上傳檔案至伺服器須符合檔案大小限制，本組訂定最大上限為 1GB。

4.2. 系統功能

本系統介面以網頁呈現，使用者以瀏覽器連上系統首頁上傳欲辨識影片後，系統即依序對影像的幀進行辨識，若定位出機車騎士與乘客，則判斷該行為者是否配戴安全帽，若無則會回傳該影像幀至前端，並附加其車牌號碼與影片的時間點，辨識完成後也提供壓縮檔案下載。圖 4-2-1 與圖 4-2-2 分別為系統首頁與辨識結果網頁。

圖 4-2-1 系統辨識首頁

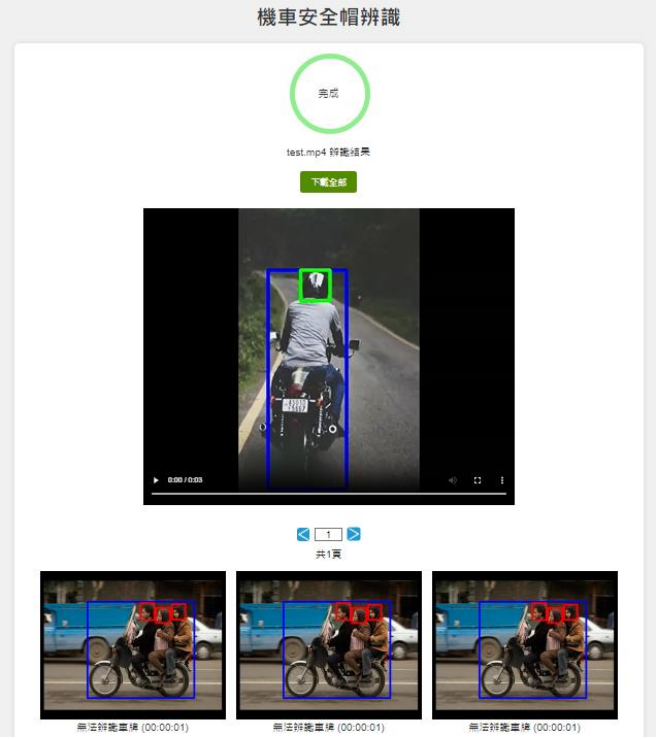

圖 4-2-2 辨識結果網頁

運用本系統的辨識功能，使用者不用再逐一檢閱整段行車紀錄影像，只需將

影像檔案上傳至系統，就能「一鍵」獲取未戴安全帽的違規事證，並進行後續檢舉作業，藉此省去人工處理的時間與精力。

第五章 結語

本組透過影像辨識技術實作了機車安全帽辨識系統，簡化違規事證的提取過程，從而節省人力與時間成本。該系統目前僅應用於民眾檢舉的處理作業，通過自動化辨識，能夠快速有效地識別未佩戴安全帽的駕駛人與乘客，為交通違規取締提供有力支援。

未來，本組期盼能以這一系統的相關技術應用於安全帽科技執法的設立，進一步擴展其應用範圍，並將模型辨識準確率提升，以更廣泛且富含實際環境的資料加入模型訓練，使其能正確辨識在不同時間、光線、情境的複雜影像，再與交通監控系統、監理單位之車牌資料庫的整合，以進行違規者的製單舉發，加強交通安全管理。再者，系統也可擴充更多安全帽違規取締以外的應用，如：以手持方式使用手機、三貼行駛的舉發等。此外，若系統的引進資料分析功能也將有助於識別高風險區域，為交通規劃與管理提供依據，並促進相關政策的制定與改進。

藉由自動化的辨識技術，能夠有效彌補人力取締無法遍及的範圍，避免因民眾僥倖心態而導致的嚴重事故，讓民眾的安全意識提升。隨著技術的持續進步，推動更為全面的交通安全措施，最終實現一個更加安全的道路環境，避免悲劇重演。

參考文獻

[1] 交通部路政及道安司。2021。未戴安全帽車禍致死率是有載安全帽者的 8 倍 不是有戴就好。168 交通安全路口網：https://168.motc.gov.tw/theme/news/post/2112201800949

[2] 內政部警政署。2003-2022。取締未戴安全帽統計。警政統計查詢網：https://ba.npa.gov.tw/statis/webMain.aspx?k=defjsp

[3] 陳宥蓁。2023。科技執法擴大執行 種類有哪些？被抓到違規罰多少？公視新聞網：https://news.pts.org.tw/article/618517

[4] 黃筱淇、紀建亨、張立陵。2023。你的安全帽呢？帶出門掛在「這」撞車易亡 嘉縣警加強查。TVBS 新聞網：https://news.tvbs.com.tw/local/2336866

[5] 林銘翰。2024。未戴安全帽等 5 違規仍可檢舉 道交條例新規定最快 6 月上路。聯合新聞網：https://udn.com/news/story/6656/7962192

[6] 張耀中、周平虎。2023。保亭：创新打造“随手拍+电子警察”治理新模式。中国警察网：https://news.cpd.com.cn/n12021581/n12021601/823/t_1097685.html

[7] ？。2022。清波 On-line(雲端) 與 Off-line(落地) 工地安全 AI 影像辨識。iDS+智慧安防雜誌：https://www.idsmag.com.tw/product/show_product_security_info.asp?secu_id=HCP016&search_security_id=40569

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. (2016). You Only Look Once: Unified, Real-Time Object Detection. IEEE Xplore: https://ieeexplore.ieee.org/document/7780460

[9] Ultralytics. (2024). Ultralytics YOLO Docs. https://docs.ultralytics.com/

[10] DS & AI Solutions. (2023). Losses and Their Weights in Yolov8 Linkedin: https://www.linkedin.com/pulse/losses-weights-yolov8-dsaisolutions-x1ggf

[11] Zongxuan Chai. (2024). Real-Time Automatic Detection of Motorcycle Hel

met Based on Improved YOLOv8 Algorithm. IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/10652610

[12] Xiangkui Jiang & Libing Pan. (2024). Helmet Wearing Detection for Electric Bike Riders Based on Improved YOLOv8. IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/10661394

[13] The Linux Foundation. (2019). Open Neural Network Exchange. ONNX | HOME: https://onnx.ai/

[14] Glenn Jocher, MatthewNoyce and Abirami Vina. (2024). ONNX Export for YOLO11. Ultralytics YOLO Docs: Models https://docs.ultralytics.com/zh/integrations/onnx/

[15] Microsoft. (2021). ONNX Runtime. https://onnxruntime.ai/

[16] Msakande, S-polly, Albertyang0, et al. (2024). ONNX and Azure Machine Learning. Microsoft: https://learn.microsoft.com/en-us/azure/machine-learning/concept-onnx?view=azureml-api-2

[17] Murat Sever & Sevda ÖğütA. (2021). Performance Study Depending on Execution Times of Various Frameworks in Machine Learning Inference. IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/9659677

[18] Amazon. (2024). What is OCR (Optical Character Recognition)? Amazon Web Services: https://aws.amazon.com/what-is/ocr/

[19] Matthill, Peters, Silex, et al. (2016). Automatic License Plate Recognition library. OpenALPR: https://github.com/openalpr/openalpr

[20] 張逸忠、李美億。2020。影像辨識實務應用：使用 C#。新北市：博碩文化。

[21] Anaconda. (2024). Miniconda. https://docs.anaconda.com/miniconda/

[22] Abhishek Singh Yadav. (2023). Helmet_Detection_Person_with_and_without_helmet. Kaggle: https://www.kaggle.com/datasets/abhi3022/helmet-detection-person-with-and-without-helmet/data

[23] npk7264. (2022). Helmet-detection. Kaggle: https://www.kaggle.com/datasets/npk7264/helmet-dataset

[24] Helmet Detection. (2022). Helmet Dataset. Roboflow: https://universe.roboflow.com/helmet-detection-xy5ky/helmet-htftb/dataset/3

[25] Object Detection HelmetsLicense. (2022). Motorcycle Helmet and License plate detection Dataset. Roboflow: https://universe.roboflow.com/object-detection-helmetslicense/motorcycle-helmet-and-license-plate-detection

[26] Moazzim Ali Bhatti. (2022). Helmet Wearing Persons. Kaggle: https://www.kaggle.com/datasets/moazzimalibhatti/helmet-wearing-persons

[27] CSGO Head Detection. (2022). Head datasets Dataset. Roboflow: https://universe.roboflow.com/csgo-head-detection/head-datasets

[28] Elven Kee, Jun Jie Chong, Zi Jie Choong, and Michael Lau. (2023). A Comparative Analysis of Cross-Validation Techniques for a Smart and Lean Pick-and-Place Solution with Deep Learning. MDPI: https://www.mdpi.com/2079-9292/12/11/2371