Car License Plate Recognition

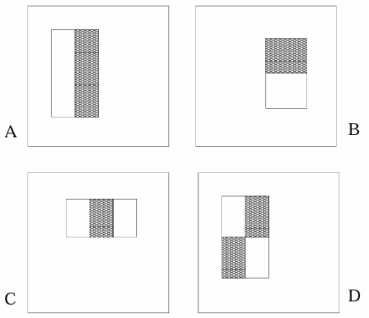
車牌辨識任務介紹：

Car License Plate Recognition ，車牌辨識是基於一種影像處理和模式辨別技術的自動化系統，通過擷取和識別車輛的車牌號碼。運作流程先由攝影機偵測當車輛進入停車場或接近閘門時，安裝的攝像機自動偵測車輛及其車牌。自動化系統可以利用無線網路介面傳送影像 ，與通過機器學習、深度學習方法應用整合來實現自動化流程。

Haar-like features

源自 Rapid Object Detection using a Boosted Cascade of Simple Features ，2001 年所出這篇論文提出三項物件辨識任務上早期的貢獻，積分影像 (Integral Image) ，可以快速計算特徵，使檢測器能夠在短時間內提取關鍵特徵，第二是基於 AdaBoost 的學習演算法 ，第三是級聯結構 ，增加分類器的複雜度 ，可快速排除背景區域 ，系統可應用於即時應用，在不依賴影像差分或膚色檢測的情況下，檢測速度達到每秒 15 幀。

物件檢測的程序中，該論文中所提及的物件檢測方法是基於**簡單特徵值**來對影像進行分類 ，特徵能夠編碼特定領域知識 ，相較於基於像素的方法，基於特徵的方法運算速度更快。 其背後靈感來源於 **Haar 基底函數**（Haar basis functions） ， 其以矩形特徵運算，相鄰區域、對角區域或針對區塊運算，設計這些特徵多樣的組合，因此也可以捕捉不同結構，且計算效率高。

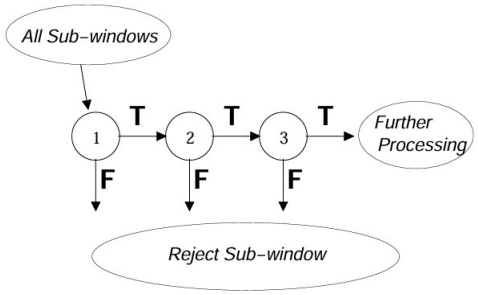


簡單來說，積分影像在特定位置存儲了**包含該點及其左上方區域**的所有像素和，利用積分影像，可以快速計算任意矩形區域內的像素和。 論文中也提到矩形特徵雖然在表達能力上相對原始，但相比於可轉向濾波器 (Steerable Filters) ，Haar 矩形特徵提供了一種**豐富的影像表示方式**，能夠支持有效學習 ，特徵雖然較粗略，但與積分影像搭配後，能夠快速運算，在效率上彌補了靈活性不足， 主要敏感於邊緣、條紋和簡單影像結構，但僅限於垂直、水平和對角線方向。

Haar Feature 其捕捉特徵的方式，上述所說的 Haar 特徵類似於**邊緣檢測算子**，用來測量影像區域的亮度變化 ，由人工所設計，透過簡單的**矩形特徵對比**，可以快速檢測影像中的邊緣、條紋和簡單結構，類似於**模板匹配**，過位置、尺度、方向的變化產生超過 180,000 個特徵 ，涵蓋了大部分常見的物件邊緣特徵，在提取特徵過程中利用積分影像 快速計算矩形特徵和，下一步後，就可以使用機器學習模型篩選出重要的特徵，減少冗於的存在， 該論文中使用 AdaBoost 篩選。

Haar- Cassacade Classifier

物件偵測中 處理效率是至關重要的。Haar 特徵偵測運用了級聯分類器 (Cascade Classifier) 來達到高準確率和低計算量的平衡。 目的是快速濾除非目標區域 ，逐層精細分類 。初步先使用簡單分類器，像是只使用兩個特徵 ，根據 AdaBoost 訓練，挑選檢測率最高的特徵 ，後續層級分類器越來越複雜且準確，增加特徵數量，進一步篩選，每一層都通過才能確認為目標物件。 因為大多數影像區塊為負樣本，在初期篩掉大部分子窗口，可以顯著減少後續計算量。



AdaBoost 訓練過程中，初期訓練中，從 180,000 多個 Haar 特徵中挑選出最具鑑別力的一小部分。 每個特徵都會進行**弱分類器訓練**，計算分類誤差。 誤差小的特徵會被賦予較高的權重。 誤分類樣本的權重會逐輪增加，使其在下一輪訓練中能夠得到更多關注。 最終將多個弱分類器**組合成強分類器**，達到較高的分類準確率。 另外也會使用 Non-Maximum Suppression 、Intersection over Union 避免**多個重疊窗口** 。

OpenCV Haar 物件偵測介紹

cv2.CascadeClassifier：用於加載訓練好的 Haar 分類器模型。、detectMultiScale：檢測圖像中的物件。

```
plates = plate_cascade.detectMultiScale(
    gray,
    scaleFactor=1.05,
    minNeighbors=8,
    minSize=(90, 30),
    maxSize=(400, 150),
)
```

控制 縮放因子、每個候選矩形至少有幾個相鄰矩形、偵測物件的最小最大尺寸進行更精確的物件偵測以此來達到車牌辨識擷取專注在車牌的任務。

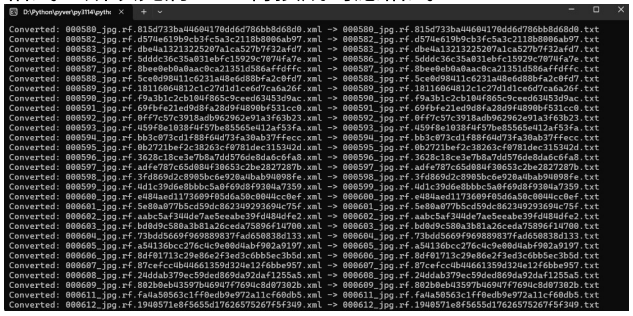
通過簡單的測試比對後，opencv 開源社群提供的 russian_plate_number .xml 實驗應用中有最好的效果，通過 opencv haarcascade 模型讀取即可實際應用。實作過程中，利用預處理影像的方式 將彩色影像轉為灰階，加速偵測過程。 根據應用場景選擇合適的 scaleFactor 和 minNeighbors。 調整窗口滑動步長和縮放比例，以在速度與精確度間取得平衡， **輕量和快速運算特性**在嵌入式系統中仍具價值 。

資料集介紹 :
Taiwan Licence Plate Recognition Research tlprr Computer Vision Project 源自於 robotflow 資料集。

Haar Feature 模型訓練建立

前面使用 Russian 的車牌辨識權重，但這可能無法泛化到諸多問題上面，例如說台灣的車牌情況可能不一定都很好分類，或是在其他情況上物件偵測如何更針對問題設計。這便是這部分實驗目標，通過建立 Haar 特徵分類器，利用資源檔案訓練我們的分類器，分成正樣本與負樣本，進行訓練。

使用前面說到的這個 tlprr 資料進行 haar cascade 模型訓練，因為資料集沒有提供合適 haar cascede 的格式，所以先將 xml 轉換成對應格式。



轉換完成後，我們現在有圖片對應 txt 檔案標註檔案，

代表 Labeled 車牌位置，以此方式讓訓練模型更去找出符合定位車牌的特徵。make positive 中將車牌圖片為正樣本，包含車牌，每個數值代表車牌數量，其車牌位置的座標左上與右下。下載 OpenCV 原始碼 ，使用 CMake 建立工程 ，CMake 建立好 opencv_traincascade.exe 與 opencv_apps 等檔案，接下來就可以正式進入訓練階段了。

Haar 訓練流程 :

positives.vec, full.vec（正樣本 .vec 檔）
負樣本圖片（灰階、無目標）

訓練會需要一個 pos.txt 檔案紀錄[圖檔路徑] [物件數量] [x y width height] ，使用轉換工具 opencv_createsamples.exe 建立 vec 檔案以便於後續訓練，所以我們使用檔案可以產生出 positive 跟 negative 的 vec 檔案，positive 用的是 TLPRR 資料，negative 用的是 kaggle scence 的資料，取 1000~4000 張，正樣本取 2800~3200 張。
“ opencv_createsamples -info pos.txt -vec positives.vec -w 24 -h 24 “ 建立 vec 方法，建立 3000 樣本數的 vec。

開始訓練 haar cascade : “ opencv_traincascade -data data -vec full.vec -bg neg.txt -numStages 15 -numPos 3200 -numNeg 8000 -w 24 -h 24 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024 “

```
PARAMETERS:
cascadeDirName: data
vecFileName: positives.vec
bgFileName: neg.txt
numPos: 2800
numNeg: 4000
numStages: 15
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : -1
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [24,24] : 162336
```

```
===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   2800 : 2800
NEG count : acceptanceRatio    4000 : 1
Precalculation time: 9.046
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
```

等待 … … …

一個階段裡面會訓練多個弱分類器（N 的數量），直到該階段達成設定的命中率（hit rate）和誤報率（false alarm rate）為止。
通常建議至少 **10 個 stage**，這是很多範例和教學推薦的數量。 可以設成 15 或更多，但訓練時間也會更長。
到 stage 4 時，NEG count : acceptanceRatio 4000 : 0.0500394，意味著分類器已經能過濾掉 95% 的負樣本了 。

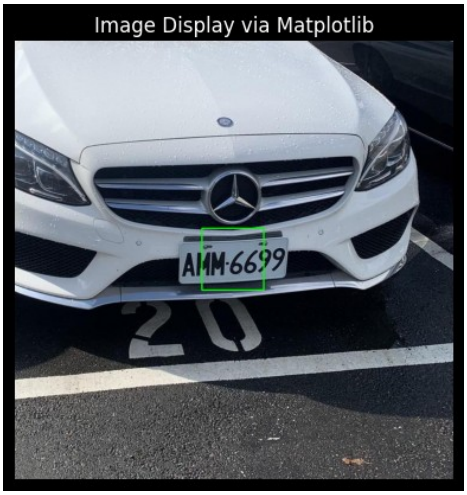
訓練時長 : 3 days 2 hours 25 minutes .
最終訓練階段 stage 15
會在 data/中保存 **cascade.xml** 為 模型的保存權重。可以過濾掉負樣本，最終只有 0.000226786 的負樣本會被誤判為正樣本。

自訂資料集訓練 Haar Cascade :

```
===== TRAINING 12-stage =====
<BEGIN
POS count : consumed   3200 : 3397
NEG count : acceptanceRatio    8000 : 0.000226786
Precalculation time: 20.354
```

===== TRAINING 14-stage =====
POS count : consumed 400 : 442
NEG count : acceptanceRatio 1000 : 5.8037e-05
迭代到 15 次時已經具備特徵辨識能力，訊連結束後也可以將 cascade.xml 權重檔案部屬至問題使用。

結果 :



如上是訓練後最終預測，結果，然而經過幾次測試其準確框出車牌範圍的能力還是太弱，使用已經訓練好的 russian 權重檔案效果更穩定，可能是訓練時哪裡做得不夠好，或是有更多提升訓練方式的空間，在後續我也將稍稍探討，所以改為使用 russian 權重檔案引入至模型更好，以下我將展示使用 russian 部屬至問題車牌辨識的效果與輸出結果。

```
plate_cascade =
cv2.CascadeClassifier("model/haarcascade
_russian_plate_number.xml")
```

新的篇章

在了解與實際應用了 Haar Feature 進行物件辨識車牌位置與擷取，Haar 模型論文原理與訓練我也帶蓋了解了，到這裡 Haar 已經告一個段落，但是呢 Haar Feature 物件辨識模型是 2001 年所出的技術，當探尋現今日物件辨識領域的 State Of The Art，已是著名基於 CNN 所設計 YOLO 系列、RCNN 或 DETR 等等模型。

YOLO 系列在眾多物件識別模型中，出色的特點就在於其具備了 CNN 帶來的高效特徵提取能力、快速的速度，由於特徵與分類的處理被一體化因此有快速的能力，在 YOLO version 3 引入了特徵金字塔 FPN 的檢測架構，弭補初代準確度的問題，因而成為今日的主流模型。

簡單談 CNN 應用

Convolutional Neural Network ，CNN 是今時電腦視覺影像辨識的主流模型，隨然相比先前各類視覺與機器學習模型，速度上 CNN 慢了不少，且吃效能，但通過 GPU 平行化運算可以緩解這個問題。卷積的圖像處理本質上來說是一種特徵工程，以往會通過類似於 Sobel、Laplace Filter 這些卷積處理來提取特徵，但面對不同問題需要設計太多 filter 來處理特徵，因此 CNN 旨在機器學習出適用的特徵提取器，同時，由於其具平移不變性與感受野機制，因此可以在特徵幾何空間上處理，並在後續應用在圖像分類、物件辨識、語意分割等任務。

You Only Look Once

YOLO 是實時處理為導向的單階段物件偵測模型 ，直接從輸入影像預測邊界框（bounding boxes）及對應的類別機率 ，YOLO 將整張影像劃分為固定大小的格子（通常為 S×S） ，每個格子預測固定數量的邊界框及其對應的置信度與類別分布 ，至今已迭代了 11 個版本與其他變體，通過端到端模式訓練與部屬，速度快速且準確，整個 YOLO 辨識模型架構包含，頭部、頸部、骨架，骨架往往描述的模型對圖片分類所用的模型，其提取高層次特徵，頸部則加強並融合多層次資訊與特徵，也就是上述提到的特徵金字塔，頭部則根據頸部傳來的訊息做最終的預測輸出，包含邊界框、每個框類別、信度，過去 YOLOv3 引入特徵融合、標定錨點為準確度帶來高效能提升，後續 v5 與 v8 在分類與定位都有很高的效能與能力，而 v8 由 ultralytics 所開發，另外，YOLOv11 是 Ultralytics 於 2024 年 9 月推出的最新一代目標檢測模型 ，具備 C3k2 、C2PSA 模組作為骨架，以路徑聚合網路 PAN 做為特徵融合，頭部則是深度可分離卷積與多尺度檢測頭 ，若看 YOLOv11 變體中的最優者，在 2025 COCO 基準評估中排行 21 名。因此，回到車牌辨識專案中，我將使用 ultralytics 的 API 來應用 YOLOv11。

應用 YOLOv11

首先，先準備好訓練資料集，在這裡可以使用先前 Haar 訓練時用的 TLPRR Dataset，不過這次改為使用 YOLO 的格式［x , y , width , height , classes］。

下載 API : pip install ultralytics
需要一個訓練配置文件 : data.yaml

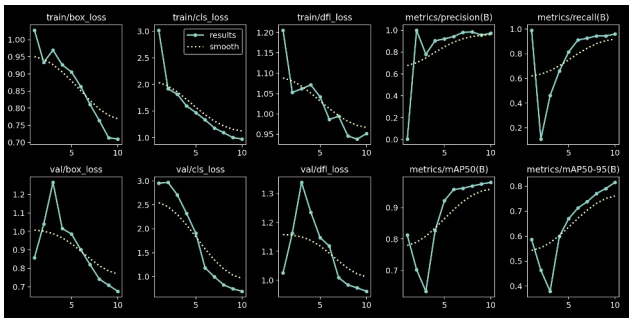
```
train: tlprr/train/images
val:   tlprr/valid/images
test:  tlprr/test/images

nc: 1
names: ['0']
```

以及對應的讀取檔案之程式。
接著，連接 API，撰寫程式使用 YOLOv11 並呼叫 train()函式，需要配置參數包含 : 訓練資料集配置檔案、訓練次數、批次大小、圖片大小。

```
model = YOLO('yolo11n.pt')
results = model.train(
    data=data_yaml,
    epochs=epochs,
    batch=batch,
    imgsz=imgsz,
    project='runs/train',
    name='tlprr_yolov11',
    exist_ok=True
)
return results
```

Epoch 迭代次數可以設為大概 10 次，並將訓練資料集檔案配置丟給 API，每個批次處理 16 張，640 pxl 圖片大小。就可以直接開始訓練，並保存最佳權重檔案，在使用 eval 驗證時便可直接讀取訓練好的權重檔案，並計算最終預測 mAP 結果，足夠好就可以直接部屬模型到問題上，對圖片進行預測與物件偵測任務。



通過使用 Ultralytics API 預訓練的模型，並遷移以 tlprr 車牌辨識資料集訓練，訓練資料只用了 338 張圖像，驗證資料使用 169 張，迭代訓練 10 次後可獲得一個可良好的用於車牌物件偵測的 YOLO 模型，mAP50 評估表現 0.98，YOLO11n 融合 100 層、2,582,347 個參數、0 個梯度、6.3 GFLOP。在這裡我用的是 v11 nano 版本，另外也可以嘗試使用 small、medium、large、x-large 等版本。

訓練與預測展示

應用辨識模型：

1. 載入訓練完的模型（.pt 檔）
2. 對指定圖片做預測
3. 把每個偵測框裁切（crop）出來
4. 儲存到指定資料夾

```
model = YOLO(model_path)
results = model.predict(image_path, save=True, save_txt=False)
```

取得框座標（格式：x1, y1, x2, y2）

```
x1, y1, x2, y2 = map(int, box.xyxy[0])
cropped = image[y1:y2, x1:x2]
```

01_crop1.jpg　02_crop1.jpg　03_crop1.jpg

完成這部分對於車牌物件辨識的擷取，使用 haar cascade 或 yolo 都可以有很好的擷取效果，接續只取車牌圖像部分，後續便可做一些辨識模型來辨識出車牌上的文字內容是甚麼，來讓電腦可以實現自動化辨識即可完成整個車牌辨識自動化 AI 機器專案。

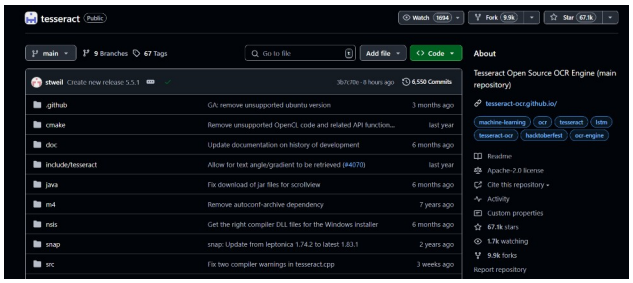
車牌 OCR 辨識器設計

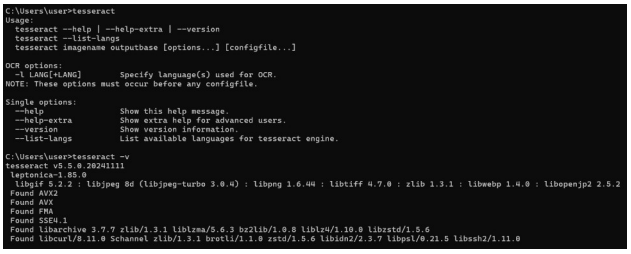
當完成使用 Haar Cascade 模型或 YOLO 模型辨識並擷取車牌物件後，我們任務是讓電腦知道車牌上的文字內容是甚麼，便需要使用一些 OCR 的技術應用。

此部分的任務目標是自動化擷取車牌文字 ，過去幾天內的實驗中，嘗試包含各項傳統方法，需要切格出圖片中每一個字元， 我嘗試了諸如去噪、二值化、形態學、自適應 + Otsu 混合，但是其實際表現對於低解析度、字元間黏連、底紋干擾極為敏感，分割錯誤率高。

OpenCV 的影響處理方法雖可偵測到文字區域，但難以保證一圖片中就是一文字，因此在我的搜索現有 OCR 的處理與模型方法後，我嘗試使用 **Tesseract OCR** 開源 模型，對自然場景文字有 非常好的辨識能力。

安裝流程 :
可以到 Tesseract github page 提供的安裝說明，以他的方式安裝，或是直接使用 https://tesseract-ocr.github.io/tessdoc/Installation.html 提供之安裝執行檔，到 Windows 處 Tesseract UB Mannheim 下載 64 bits 執行檔案下載，並安裝，安裝後再本機電腦中將 tesseract-OCR 放到系統環境變數 path 當中。





Tesseract OCR 模型使用

--pip install pytesseract opencv-python
設定方式:
`--psm 7`：Treat the image as a single text line
tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

config = r'-c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 --psm 7'

```
from PIL import Image
import pytesseract
print(pytesseract.image_to_string(Image.open('test/license_plate.jpg')))
```

在原版最簡單的使用中會出現一些問題，像是使用辨識由於放入輸入圖像不太清晰，模糊等情況，導致她會出現一些字體辨識問題，甚至可能完全無法辨識的結果產生。為解決這個問題，我決定在輸入圖像上加入一些預先處理與影像調整的功能。

雜訊誤判 :
　　可以通過加入正則表達提取需要的內容 r'[^A-Z0-9]'，同時，規範預測內容只允許 `A-Z`、`0-9`。

傾斜視角 :
　　Tesseract 無法自動校正強烈斜視，導致辨識率下降 ，所以我使用解決方法，先行透視校正，邊緣偵測 與 最大四邊形擬合 。

```
    # TTarget Coordinate
    dst = np.array([[0,0], [maxW-1,0], [maxW-1,maxH-1], [0,maxH-1]], dtype="float32")
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(img, M, (maxW, maxH))
    return warped
```

字元混淆 :
自訂 whitelist ，後處理時，依車牌格式用正則進行微調，如台灣「1~2 英字 + 4 數字」→ re.search(r'[A-Z]{1,2}[0-9]{4}', text)

強化圖像 :
增加了簡單的圖像處理功能，先對輸入圖像 reshape 放大多倍，使用 cv2 CLAHE 局部對比度，以及加入銳化圖像操作使用高斯方法。

為何選擇 Tesseract OCR 而非傳統切割+CNN 。以實驗中的穩定度來看，傳統切割易受解析度、字距、背景干擾影響 ，在我使用的資料集與複雜多樣圖像中，容易出現問題與甚至無法提取切割單一字元圖像等情況，改用 tesseract 可以很好解決這個問題。
Tesseract OCR 在大多數常見車牌場景下，皆能以最低開發成本，達到優於傳統字元切割 + CNN 的辨識效果。

Tesseract OCR 實驗結果與程式碼：

結合前面上述提到的所有影像處理方案與使用 Tesseract OCR 圖像轉文字辨識模型程式碼如下

實驗圖片[**Haar Cascade 擷取部分**]：

原版 tesseract OCR 預測：EPGeMN112

畫質強化圖像：

```python
# 1. 畫質強化
def enhance_image(img, scale=2):
    # 1.1 放大
    h, w = img.shape[:2]
    img = cv2.resize(img, (w*scale, h*scale), interpolation=cv2.INTER_CUBIC)
    # 1.2 CLAHE (局部對比度)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    gray = clahe.apply(gray)
    # 1.3 銳化 (Unsharp Mask)
    blur = cv2.GaussianBlur(gray, (0,0), sigmaX=3)
    sharp = cv2.addWeighted(gray, 1.5, blur, -0.5, 0)
    return sharp
```

圖片視角斜向處理：

```python
# 2. 斜視透視校正
def rectify_plate(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    cnts, _ = cv2.findContours(edges, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
    for c in cnts:
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
        if len(approx) == 4:
            pts = approx.reshape(4, 2)
            break
    else:
        return img

    rect = np.zeros((4,2), dtype="float32") # 依左上、右上、右下、左下排序
    s = pts.sum(axis=1); rect[0] = pts[np.argmin(s)]; rect[2] = pts[np.argmax(s)]
    diff = np.diff(pts, axis=1); rect[1] = pts[np.argmin(diff)]; rect[3] = pts[np.argmax(diff)]

    # 計算寬高
    (tl, tr, br, bl) = rect
    widthA = np.linalg.norm(br - bl)
    widthB = np.linalg.norm(tr - tl)
    maxW = int(max(widthA, widthB))
    heightA = np.linalg.norm(tr - br)
    heightB = np.linalg.norm(tl - bl)
    maxH = int(max(heightA, heightB))

    # TTarget Coordinate
    dst = np.array([[0,0], [maxW-1,0], [maxW-1,maxH-1], [0,maxH-1]], dtype="float32")
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(img, M, (maxW, maxH))
    return warped
```

Tesseract OCR：

```python
#  OCR Prediction
def ocr_plate(img):
    pil = Image.fromarray(img) # PIL 轉灰階
    config = r'-c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 --psm 7'
    text = pytesseract.image_to_string(pil, config=config)
    text = re.sub(r'[^A-Z0-9]', '', text) # 移除所有非英數
    return text
```

最後辨識結果
辨識字串： PGMN112 。

可以發現到 多出來的 ‘E’ 與 ‘e’ 不會誤判。
另外再測試傾斜視角圖片時也可以看到原本完全無法偵測回傳空字串變成可以正常預測但準度並非是百分之百，可能會有些許部分錯誤。

車牌偵測與辨識小專題整合

在這一部分中，我在整合兩個大部分實驗，以 **Haar Cascade 、YOLOv11 車牌偵測** 與 **Tesseract OCR** 完整整合成一支 End-to-End 腳本：輸入原始影像 ，物件偵測並標記框框 ， 擷取車牌 ROI ，透視校正 ，畫質強化 ，OCR 辨識 ，在原圖繪製辨識結果與文字。

整合專案預測辨識車牌測試：

上面這張圖是 Haar + Tesseract 辨識結果。
在使用 YOLO 時就有點不一樣了，我發現如果用了剛剛那張圖片會辨識不到車牌可能是因為訓練太少，而且使用 YOLO 後不能再使用圖像強化或傾斜處理部分，反而會影響結果，這背後邏輯暫時我還不清楚。

YOLOv11 + Tesseract ：

整體來說，已經算是實現整個 pipeline 流程，並且可以達到很不錯的效果了，後續也可能嘗試更多來達到更好效果。此專案我也將其開源在 Github 上，詳細的程式碼與開發方式也都寫在檔案內。
My Repo : https://github.com/JiaenSuen/License-Plate-Recognition。
未來展望：
採用更輕量且精準的 物件偵測模型，實現 30 FPS 以上的即時監控系統。 擴大資料集多樣性 ，並且可以後續部署到邊緣裝置，或是識別部分，可基於 PyTorch 建立端到端車牌辨識網路 ，自訂 CRNN 與 CTC 模型，或多元 OCR 引擎整合，進一步降低單一模型錯誤率 ，可以後續通訊模組的智慧攝像頭，將辨識結果與時戳、GPS 座標打包上傳至雲端 ，結合場內道閘控制與車牌辨識系統，自動放行已註冊車輛，並可透過手機 App 推播停車資訊及付款訊息。