|  |  |  |  |
| --- | --- | --- | --- |
| **影像處理Image Processing**  **Final Project - License Plate Recognition** | | | |
| **隊名** | **DIP** | | |
| **職稱** | **學號** | **姓名** | **負責工作** |
| **專案負責人** | **11077002** | **蘇孝芳** | **程式撰寫 / 書面報告** |
| **組員** | **11077609** | **葉欲寬** | **程式撰寫** |
| **組員** | **11077604** | **馮立文** | **資料搜尋與提供 / 書面報告** |
| **聯絡電話** | **0979689211 / 0926956537(優先)** | | |
| **指導教授: 張元翔** | | | |

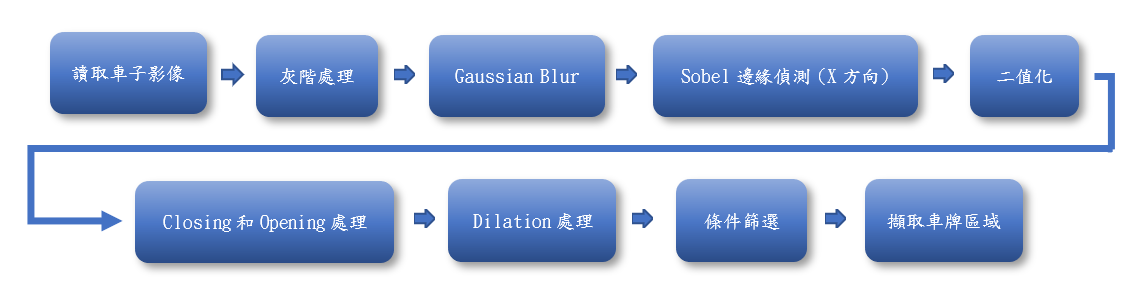
**I.Introduction**

汽車已經成為現今的主要交通工具，而車牌就猶如汽車的身份證，想要對汽車進行有效的管理從車牌著手最為實際，本篇專題主要是針對汽車的車牌，做相關的影像處理及辨識的實作，進而可以應用到大樓的門禁系統、停車場的管理系統，甚至是贓車的查詢系統。此專題辨識流程分為兩大部份，第一部份是將影像中的車牌區域擷取出來，以供後續辨識使用；第二個部份是將前面擷取出的車牌影像中的號碼依序切割出來並加以辨識，最後將辨識結果顯現出來。

**II. Method**

**1.車牌擷取**

先將車牌轉成灰階影像，使用Gaussian Blur減少影像的雜訊，使能更好地找出更精準的輪廓，然後使用Sobel做邊緣偵測，計算x方向的梯度(不計算y方向梯度)，原因是車牌上的數字在豎方向較長，重點在於得到豎方向的邊界，留下明顯的垂直邊緣後接著進行二值化，再利用形態學的Closing和Opening，先Closing將車牌數字部分連線，再Opening將不是塊狀的或是較小的部分去掉，由於部分影象得到的輪廓邊緣不整齊，因此再進行一次Dilation，此時會得到一些區塊(含：車牌區塊、非車牌區塊)，最後使用條件篩選(如：車牌的寬高比)將車牌區塊擷取出來。



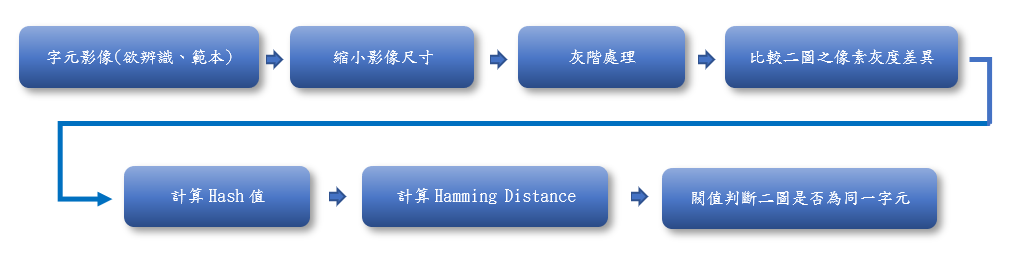
**2.字元分割**

對二質化的車牌影像(此處影像為黑底白字)計算其每列黑、白像素總和，然後給定參數做為要分割字元的開始位，分割後會留下多個影像寬度為字元寬度的切割字元影像(字元左右多餘黑邊切掉)，高度維持原樣(字元上下多餘黑邊仍在)。

**3.字元比對**

此處使用perceptual hash algorithm做字元比對。先計算兩張圖片的Hash值，再透過Hash值計算兩張圖片的Hamming Distance，依Hamming Distance的大小來判斷兩張圖片的相似程度。Hamming Distance越大代表2張圖相似度越低，達到指定閥值就將此”欲辨識的字元”判別為該”範本字元”。

在計算Hash值前先將影像縮小尺寸到8\*8，可去除影像的細節，透過保留結構/明暗等基本信息，來減少不同影像比例所產生的差異，然後影像灰度化以減少計算量(會更快)，接著比較像素的灰度，若第一個影像之像素的顏色強度大於第二個影像之像素，差異值設True（1），若非則設爲False（0），再將差異值數組中每個值看做一bit，每8 bit組爲一16進制值，將16進制值連接起來轉換爲字符串，得出Hash值。透過比對2個影像(欲辨識的字元、範本字元)，查看64位(8\*8)中有多少位是不一樣的，得出Hamming distance，Hamming Distance越小代表2張圖越相似，達到指定閥值就將此”欲辨識的字元”判別為該”範本字元”(2張圖為同一字元)。



**III. Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Original Image** | | **Gray Image** | |
|  | |  | |
| **GaussianBlur Image** | **Sobel Image** | | **Binary Image** |
|  |  | |  |
| **Closed Image** | **Opened Image** | | **Dilation Image** |
|  |  | |  |

**Fig. 1** *Image preprocess*

|  |  |  |
| --- | --- | --- |
| **Original Image** | **License Plate Detection** | **Binary Image** |
| (Train01.jpg) |  |  |
| (Train02.jpg) |  |  |
| (Train03.jpg) |  |  |
| (Train04.jpg) |  |  |
| (Train05.jpg) |  |  |

**Fig. 2** *License Plate Detection* & *Binary Image of License Plate*

|  |  |
| --- | --- |
| **Binary License Plate Image** | **Split character from Binary License Plate Image** |
| (Train01.jpg) |  |
| (Train02.jpg) |  |
| (Train03.jpg) |  |
| (Train04.jpg) |  |
| (Train05.jpg) |  |

**Fig. 3** *Split character from Binary License Plate Image*

|  |  |
| --- | --- |
| **License Plate Image** | **License Plate Recognition Result** |
| (Train01.jpg) |  |
| (Train02.jpg) |  |
| (Train03.jpg) |  |
| (Train04.jpg) |  |
| (Train05.jpg) |  |

**Fig. 4** *License Plate Recognition Result*

**IV. Discussion**

**1.車牌擷取**

若車子都是有色(非白色)的話，車牌擷取可以使用顏色偵測的方式找出車牌位置，將除車牌外的部分遮罩掉，即可僅留下所需的車牌資訊，但現實中有許多白車，純粹偵測白色此法不可行。

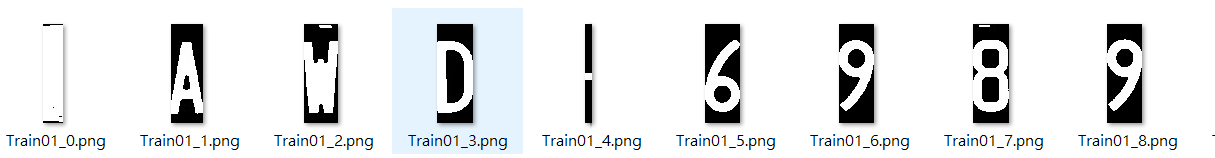
因此在本專題中，先將車子影像轉成灰階影像，並利用遮罩留下較為明顯的垂直邊緣，接著進行二值化，再經過膨脹處理後，使得車牌區域連結成一塊，再對影像做侵蝕處理以進行篩選，清除細小的非車牌區塊，最後經過標籤化及條件篩選將車牌區塊擷取出來。

理想中的車牌擷取成果為只擷取車牌本體，甚至最好切掉車牌上緣與下緣多餘白邊的部分，尤其是上緣，因上緣通常包含車牌螺絲、倒車攝影和自然光造成的陰影等因素，這些因素與之後的字元切割成效息息相關，若能在做字元切割的前期處理中就解決是最理想的。

**2.字元切割**

在做字元分割前，會先對切下來的車牌做二質化處理，須妥善選擇閥值以妥善將車牌白底部分與字元區隔開。在做處理時，會發現光線所造成的陰影、反光，甚至是車牌螺絲、倒車攝影等都會影響二質化後的影像成效，若處理不善會影響到字元的邊緣偵測，進而影響到字元分割的成果。

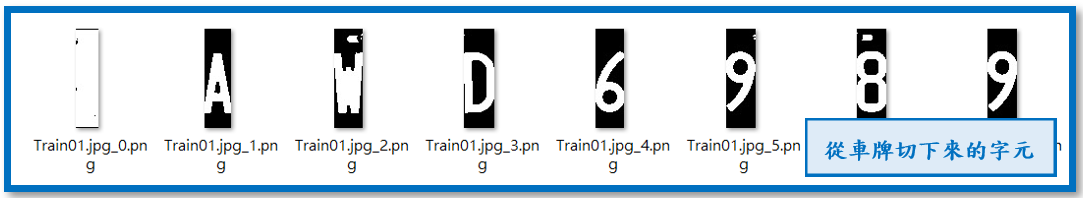
 

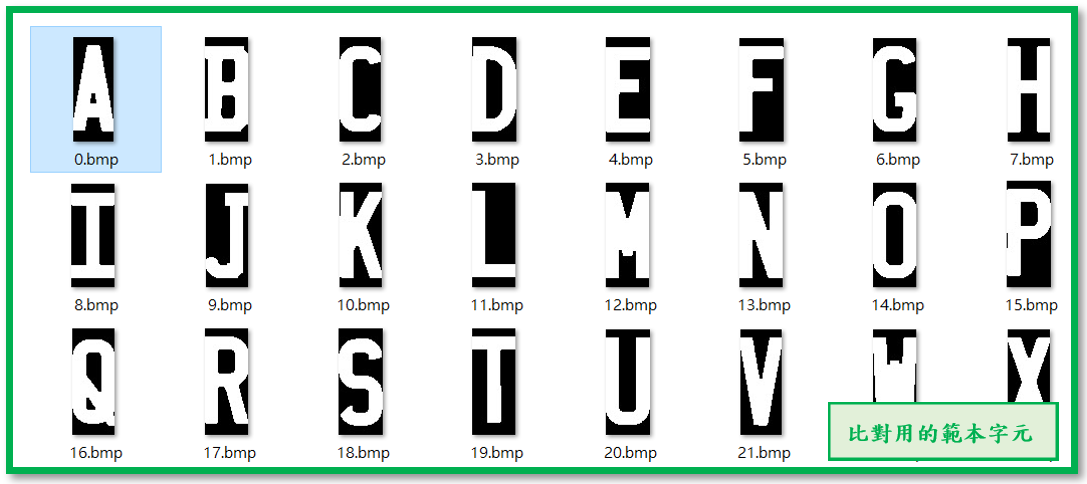


除車牌外的多餘背景經過二質化、字元切割的處理，會出現也被切分為一個字的情況，但只要經過字元比對，在比對過程中其會因為找不到相似的字元，因此會被排除成功，不會判別為其他字元。

**3.字元比對**

使用perceptual hash algorithm的成效受字元切割之成果影響極大，因此方法是使用影像間的像素灰度之差異來判別相似與否，若用來比對的字元影像中的多餘部分(除了字元以外的部分)比例不一，會使其之間的相似度降低，影響比對結果。





從**Fig. 4**的結果來看，字元比對方面仍有許多需要改善，形狀較類似的字元容易誤判，這需要透過改善比對方式或嘗試使用其他字元比對方法，來提升字元辨識成效。



**VI. Appendix**

|  |
| --- |
| Python 程式 |
| # 影像處理  # 期末專題 Final Project  # 學號: 11077002 / 11077604 / 11077609  # 姓名: 蘇孝芳 / 馮立文 / 葉欲寬  **# detect\_car\_license.py 偵測車牌用**  import cv2  import numpy as np  def preprocess(img):  # 前處理：包括灰度處理，高斯濾波平滑處理，Sobel提取邊界，影象二值化  # 對於Sobel提取邊界的引數設定，第四個引數設為零，表示不計算y方向的梯度，原因是車牌上的數字在豎方向較長，重點在於得到豎方向的邊界  # 對於二值化函式的引數設定，第二個引數設為220，是二值化的閾值，是一個經驗值  gray\_img = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)  kernel\_size = 5  GaussianBlur\_img = cv2.GaussianBlur(gray\_img, (kernel\_size, kernel\_size), 0)  # cv2.imshow("GaussianBlur\_img", GaussianBlur\_img)  # cv2.waitKey(0)  Sobel\_img = cv2.Sobel(GaussianBlur\_img, -1, 1, 0, ksize=3)  # cv2.imshow("Sobel\_img", Sobel\_img)  # cv2.waitKey(0)  ret, binary\_img = cv2.threshold(Sobel\_img, 200, 255, cv2.THRESH\_BINARY)  # cv2.imshow("binary\_img", binary\_img)  # cv2.waitKey(0)  # 形態學運算  kernel = np.ones((40, 80), np.uint8)  # 先閉運算將車牌數字部分連線，再開運算將不是塊狀的或是較小的部分去掉  close\_img = cv2.morphologyEx(binary\_img, cv2.MORPH\_CLOSE, kernel)  open\_img = cv2.morphologyEx(close\_img, cv2.MORPH\_OPEN, kernel)  # cv2.imshow("open\_img", close\_img)  # cv2.waitKey(0)  # kernel2 = np.ones((10, 10), np.uint8)  # open\_img2 = cv2.morphologyEx(open\_img, cv2.MORPH\_OPEN, kernel2)  # 由於部分影象得到的輪廓邊緣不整齊，因此再進行一次膨脹操作  element = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))  dilation\_img = cv2.dilate(open\_img, element, iterations=3)  for x in range(280):  for y in range(1400):  dilation\_img[x, y] = 0  # cv2.imshow("dilation\_img", dilation\_img)  # cv2.waitKey(0)  return dilation\_img  def dtc\_cat\_lic(gray\_img):  # 獲取輪廓  contours, hierarchy = cv2.findContours(gray\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  # 測試邊框識別結果  # cv2.drawContours(gray\_img, contours, -1, (0, 0, 255), 3)  temp\_contours = []  for contour in contours:  if cv2.contourArea(contour) > 11000 and cv2.contourArea(contour) < 30000:  temp\_contours.append(contour)  car\_plates = []  for temp\_contour in temp\_contours:  rect\_tupple = cv2.minAreaRect(temp\_contour)  rect\_width, rect\_height = rect\_tupple[1]  if rect\_width < rect\_height:  rect\_width, rect\_height = rect\_height, rect\_width  aspect\_ratio = rect\_width / rect\_height  # 車牌正常情況下寬高比在2 - 4之間  if aspect\_ratio > 2 and aspect\_ratio < 4:  car\_plates.append(temp\_contour)  rect\_vertices = cv2.boxPoints(rect\_tupple)  rect\_vertices = np.int0(rect\_vertices)  return car\_plates  def capture\_car\_lic(img, car\_plates):  for car\_plate in car\_plates:  row\_min, col\_min = np.min(car\_plate[:, 0, :], axis=0)  row\_max, col\_max = np.max(car\_plate[:, 0, :], axis=0)  row\_min -= 15  col\_min -= 10  row\_max += 8  col\_max += 3  cv2.rectangle(img, (row\_min, col\_min), (row\_max, col\_max), (0, 255, 0), 2)  # cv2.imwrite("./dtc\_img/" + d, img)  # cv2.imshow("img", img)  # cv2.waitKey(0)  car\_license = img[col\_min:col\_max, row\_min:row\_max, :]  return car\_license  def main(filename):  img = cv2.imread(filename)  # 修改圖片大小  resize\_h = 800  height = img.shape[0]  scale = img.shape[1] / float(height)  img = cv2.resize(img, (int(scale \* resize\_h), resize\_h))  gray\_img = preprocess(img)  car\_plates = dtc\_cat\_lic(gray\_img)  # print(filename, len(car\_plates))  if len(car\_plates) == 1:  car\_license = capture\_car\_lic(img, car\_plates)  # cv2.imwrite("./car\_license\_img/" + d, car\_license)  # cv2.imshow("car\_license\_img.jpg", car\_license)  # cv2.waitKey(0)  return car\_license  if \_\_name\_\_ == '\_\_main\_\_':  data = ["Train01.jpg", "Train02.jpg", "Train03.jpg", "Train04.jpg", "Train05.jpg",  "Train06.jpg", "Train07.jpg", "Train08.jpg", "Train09.jpg", "Train10.jpg",  "Train11.jpg", "Train12.jpg", "Train13.jpg", "Train14.jpg",  "Train15.jpg", "Train16.jpg", "Train17.jpg"]  # data = ["Train17.jpg", "Train12.jpg", "Train15.jpg"]  for d in data:  main("./img/" + d)  # main("./img/Train01.jpg")  **# car\_lic\_split.py 分割字元用**  import cv2  def car\_lic\_split(img):  # 參數  binary\_threshold = 90  segmentation\_spacing = 0.93  # 前處理：灰階、二值化  resize\_h = 100  height = img.shape[0]  scale = img.shape[1] / float(height)  img = cv2.resize(img, (int(scale \* resize\_h), resize\_h))  img\_gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)  # cv2.imshow('gray', img\_gray)  # cv2.waitKey(0)  img\_thre = img\_gray  cv2.threshold(img\_gray, binary\_threshold, 255, cv2.THRESH\_BINARY\_INV, img\_thre)  # cv2.imshow('threshold', img\_thre)  # cv2.waitKey(0)  # 分割字符  white = [] # 記錄每一列的白色像素總和  black = [] # 記錄每一列的黑色像素總和  height, width = img\_thre.shape[:2]  white\_max = 0 # 僅保存每列，取列中白色最多的像素總數  black\_max = 0 # 僅保存每列，取列中黑色最多的像素總數  # 循環計算每一列的黑白色像素總和  for i in range(width):  w\_count = 0 # 這一列白色總數  b\_count = 0 # 這一列黑色總數  for j in range(height):  if img\_thre[j][i] == 255:  w\_count += 1  else:  b\_count += 1  white\_max = max(white\_max, w\_count)  black\_max = max(black\_max, b\_count)  white.append(w\_count)  black.append(b\_count)  # False表示白底黑字；True表示黑底白字  arg = black\_max > white\_max  # 分割圖像，給定參數爲要分割字符的開始位  def find\_end(start\_):  end\_ = start\_ + 1  for m in range(start\_+1, width - 1):  if(black[m] if arg else white[m]) > (segmentation\_spacing \* black\_max if arg else segmentation\_spacing \* white\_max):  end\_ = m  break  return end\_  n = 1  count = 0  char\_imgs = []  while n < width - 1:  n += 1  if(white[n] if arg else black[n]) > ((1 - segmentation\_spacing) \* white\_max if arg else (1 - segmentation\_spacing) \* black\_max):  # 上面這些判斷用來辨別是白底黑字還是黑底白字  start = n  end = find\_end(start)  n = end  if end - start > 5:  # print(start, end)  cj = img\_thre[1:height, start:end]  x, y = cj.shape  bk = 0  wt = 0  # 遍历二值图，为0则bk+1，否则wt+1  for i in range(x):  for j in range(y):  if cj[i, j] == 0:  bk += 1  else:  wt += 1  rate1 = round(wt / (x \* y) \* 100, 2)  rate2 = round(bk / (x \* y) \* 100, 2)  if count in [0, 9] and rate1 > 89:  # print("{}白色占比: {}".format((d[:7]+"\_"+str(count)), (str(rate1) + '%')))  continue  if count in [0, 3, 4] and rate2 > 89:  # print("{}黑色占比: {}".format((d[:7]+"\_"+str(count)), (str(rate2) + '%')))  continue  else:  # cv2.imwrite('./car\_lic\_char\_img/{0}.png'.format(n), cj)  # cv2.imwrite('./tmp2/{0}.bmp'.format(ii\*5+count), cj)  # cv2.imshow('cutChar', cj)  # cv2.waitKey(0)  count += 1  char\_imgs.append(cj)  return char\_imgs  if \_\_name\_\_ == '\_\_main\_\_':  # ii = 7  # for ii in range(6):  # img\_path = './tmp/{}.bmp'.format(ii)  # car\_lic\_split(img\_path)  data = ["Train01.jpg", "Train05.jpg", "Train09.jpg", "Train13.jpg", "Train17.jpg",  "Train02.jpg", "Train06.jpg", "Train10.jpg", "Train14.jpg",  "Train03.jpg", "Train07.jpg", "Train11.jpg", "Train15.jpg",  "Train04.jpg", "Train08.jpg", "Train12.jpg", "Train16.jpg"]  for d in data:  img\_path = './car\_license\_img/'+d  img = cv2.imread(img\_path)  car\_lic\_split(img)  # img\_path = './car\_license\_img/Train02.jpg'  # car\_lic\_split(img\_path)  **# DIP11077002.py 主程式**  import detect\_car\_license  import sys  import car\_lic\_split  import cv2  mapping = {  "0": "A",  "1": "B",  "2": "C",  "3": "D",  "4": "E",  "5": "F",  "6": "G",  "7": "H",  "8": "I",  "9": "J",  "10": "K",  "11": "L",  "12": "M",  "13": "N",  "14": "O",  "15": "P",  "16": "Q",  "17": "R",  "18": "S",  "19": "T",  "20": "U",  "21": "V",  "22": "W",  "23": "X",  "24": "Y",  "25": "Z",  "30": "1",  "31": "2",  "32": "3",  "33": "4",  "34": "5",  "35": "6",  "36": "7",  "37": "8",  "38": "9",  "39": "0",  }  # 均值哈希算法  def aHash(img):  img = cv2.resize(img, (9, 8), interpolation=cv2.INTER\_CUBIC)  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  s = 0  ahash\_str = ''  for i in range(8): # 遍歷累加求像素和  for j in range(8):  s = s + gray[i, j]  avg = s / 64 # 求平均灰度  for i in range(8): # 灰度大於平均值爲1相反爲0生成圖片的hash值  for j in range(8):  if gray[i, j] > avg:  ahash\_str = ahash\_str + '1'  else:  ahash\_str = ahash\_str + '0'  return ahash\_str  def cmpHash(hash1, hash2): # Hash值對比  n = 0  if len(hash1) != len(hash2): # hash長度不同則返回-1代表傳參出錯  return -1  for i in range(len(hash1)): # 遍歷判斷  if hash1[i] != hash2[i]: # 不相等則n計數+1，n最終爲相似度  n = n + 1  return n  def OCR(char\_imgs):  try:  car\_license = ""  normal = True  start = 0  end = 40  dash = 3  char\_imgs\_len = len(char\_imgs)  # print(char\_imgs\_len)  if char\_imgs\_len > 8:  normal = False  for i, char\_img in enumerate(char\_imgs):  char\_img = cv2.cvtColor(char\_img, cv2.COLOR\_BGR2RGB)  # cv2.imshow('cutChar', char\_img)  # cv2.waitKey(0)  result = {}  if normal:  if i <= 2:  start, end = (0, 26)  dash = 3  else:  start, end = (30, 40)  else:  if i <= 3:  start, end = (0, 26)  dash = 4  else:  start, end = (30, 40)  for idx in range(start, end):  img2 = cv2.imread('./template\_fonts/{}.bmp'.format(idx))  hash1 = aHash(char\_img)  hash2 = aHash(img2)  n = cmpHash(hash1, hash2)  result[idx] = n  idx = str(min(result, key=result.get))  # print(idx, ": ", result[int(idx)])  if idx in ["26", "27", "28", "29"]:  continue  car\_license\_char = str(mapping[idx])  car\_license += car\_license\_char  car\_license\_list = list(car\_license)  car\_license\_list.insert(3, '-')  car\_license = ''.join(car\_license\_list)  # print("License Plate-{}: {}".format(d, car\_license))  return car\_license  except Exception as e:  print(e)  def main(filename):  car\_license = detect\_car\_license.main(filename)  char\_imgs = car\_lic\_split.car\_lic\_split(car\_license)  car\_license = OCR(char\_imgs)  output = "License Plate: " + car\_license  print(output)  return output  if \_\_name\_\_ == '\_\_main\_\_':  # python ./img/Train01.jpg  filename = "./img/Train01.jpg"  if len(sys.argv) > 1:  main(sys.argv[1])  else:  data = ["Train01.jpg", "Train02.jpg", "Train03.jpg", "Train04.jpg", "Train05.jpg",  "Train06.jpg", "Train07.jpg", "Train08.jpg", "Train09.jpg", "Train10.jpg",  "Train11.jpg", "Train12.jpg", "Train13.jpg", "Train14.jpg",  "Train15.jpg", "Train16.jpg", "Train17.jpg"]  for d in data:  main("./img/" + d) |