第3組 SUDOKU ANSWER

110502018 范家齊

目標

分析含有Sudoku內容的螢幕截圖、拍攝照片等，並給出答案

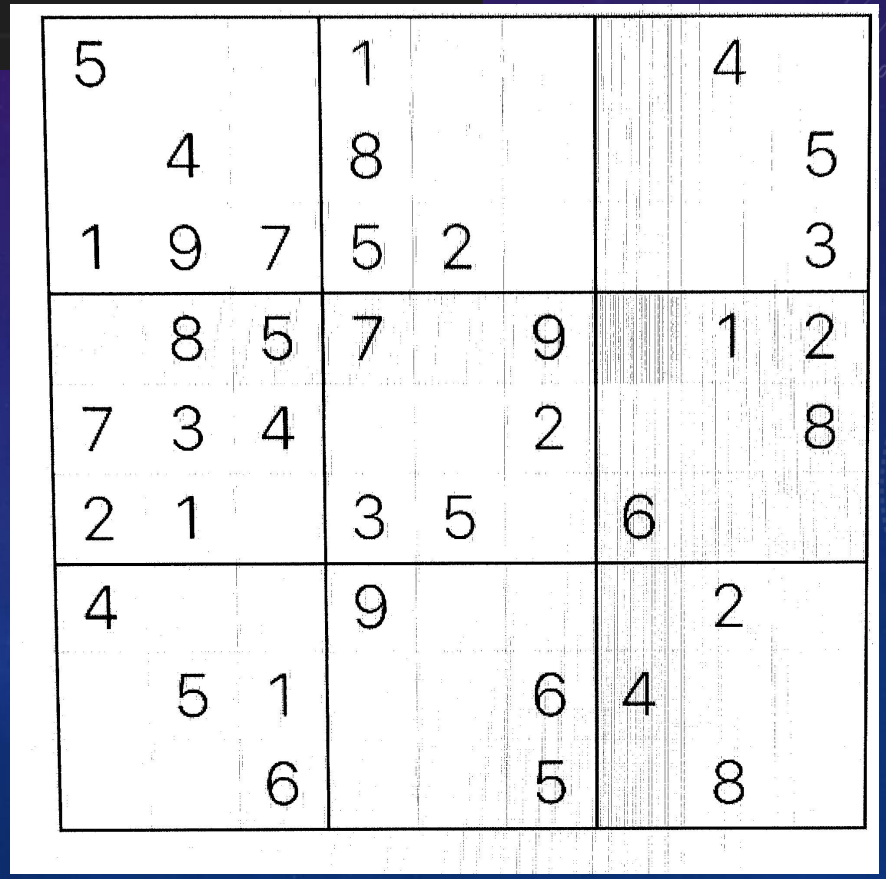
步驟

1.讀取含有Sudoku內容之照片
2.對圖片進行二值化
3.去除照片中邊框
4.切割數字
5.計算數字對應位置
6.對有數字的圖片識別
8.將識別結果製作成9*9陣列
9.計算答案
10.輸出結果 or 識別錯誤

5			1				4	
	4		8					5
1	9	7	5	2				3
	8	5	7		9		1	2
7	3	4			2			8
2	1		3	5		6		
4			9				2	
	5	1			6	4		
		6			5		8	

二值化

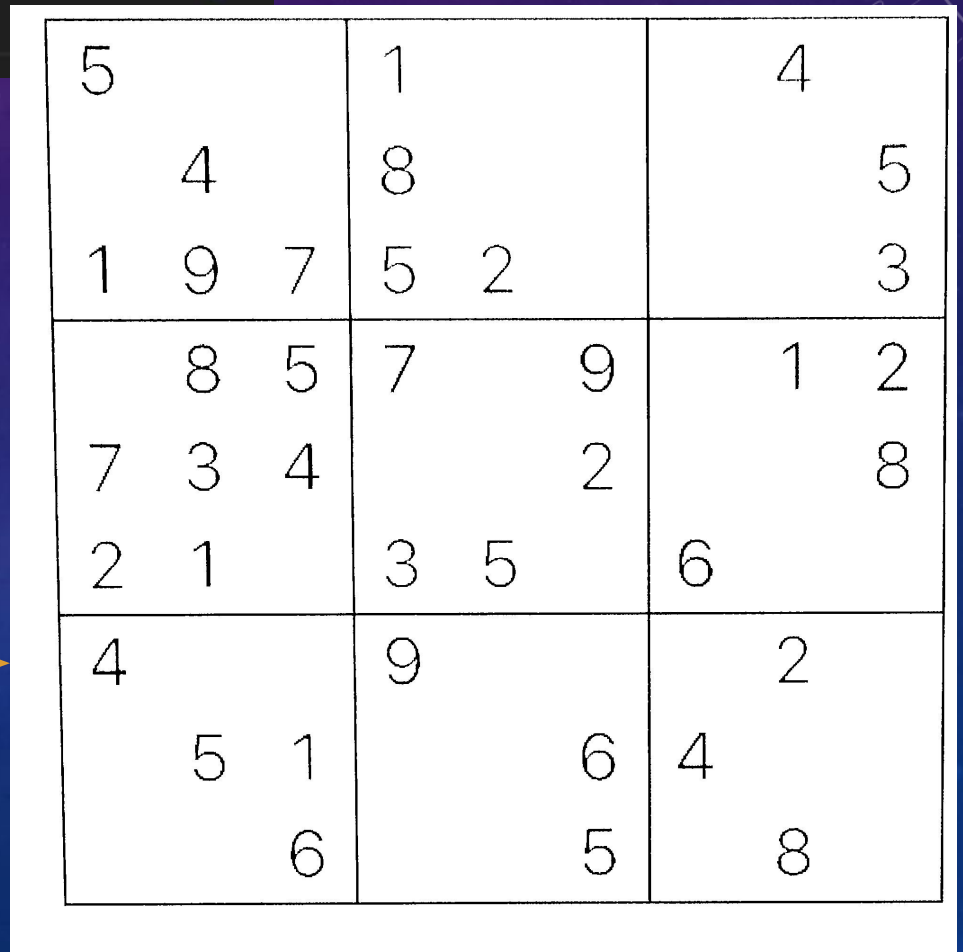
```python
#指定二值化閾值
threshold_value = 128

#二值化
ret, binary_image = cv2.threshold(resized_image, threshold_value, 255, cv2.THRESH_BINARY)
cv2.imwrite("binary_image.jpg", binary_image)
```



膨脹

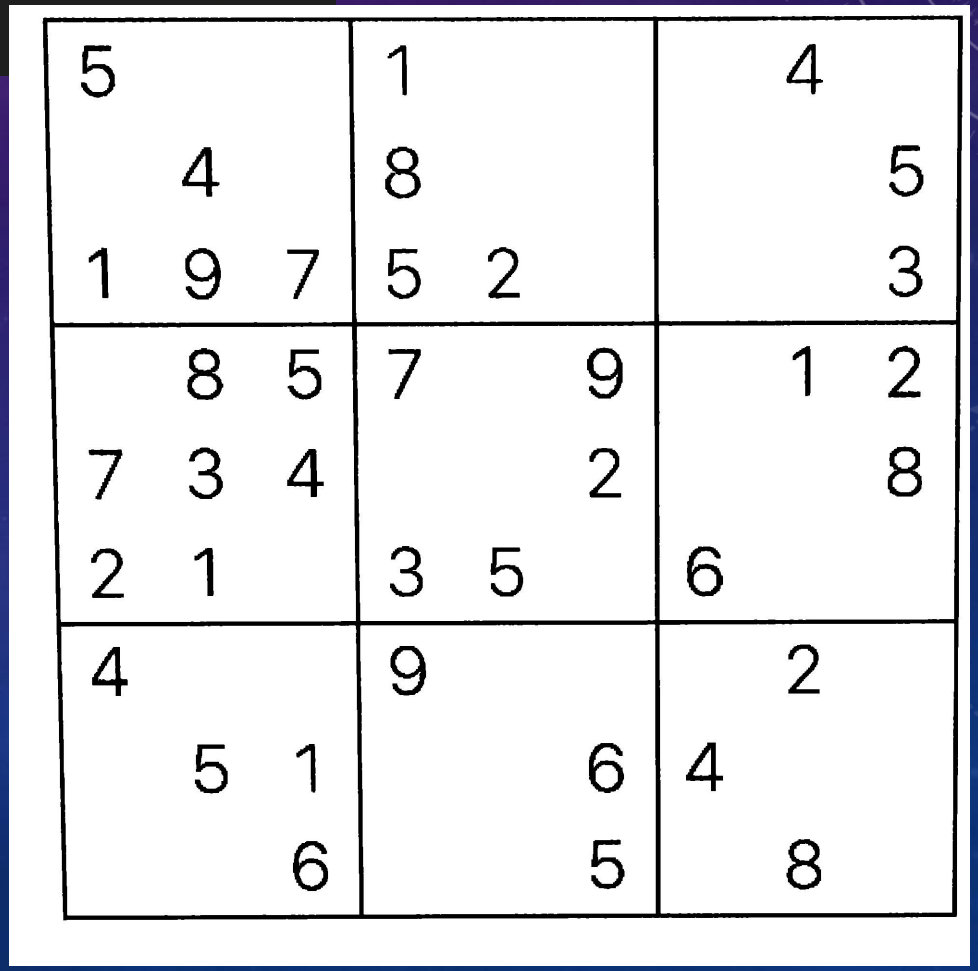
```python
#進行膨脹
kernel_size = (3, 3)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernel_size)
dilated_image = cv2.dilate(binary_image, kernel, iterations=1)
cv2.imwrite("dilate_image.jpg", dilated_image)
```

去除雜訊 →



侵蝕

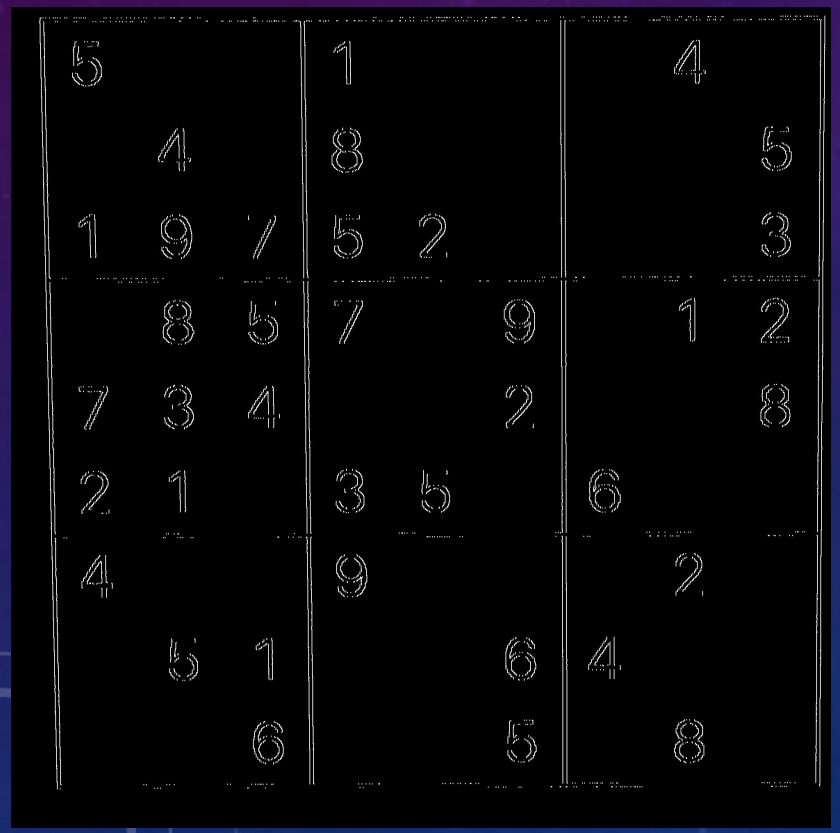
```python
#進行侵蝕
kernel_size = (5, 5)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernel_size)
erode_image = cv2.erode(dilated_image, kernel, iterations=1)
cv2.imwrite("erode_image.jpg", erode_image)
```



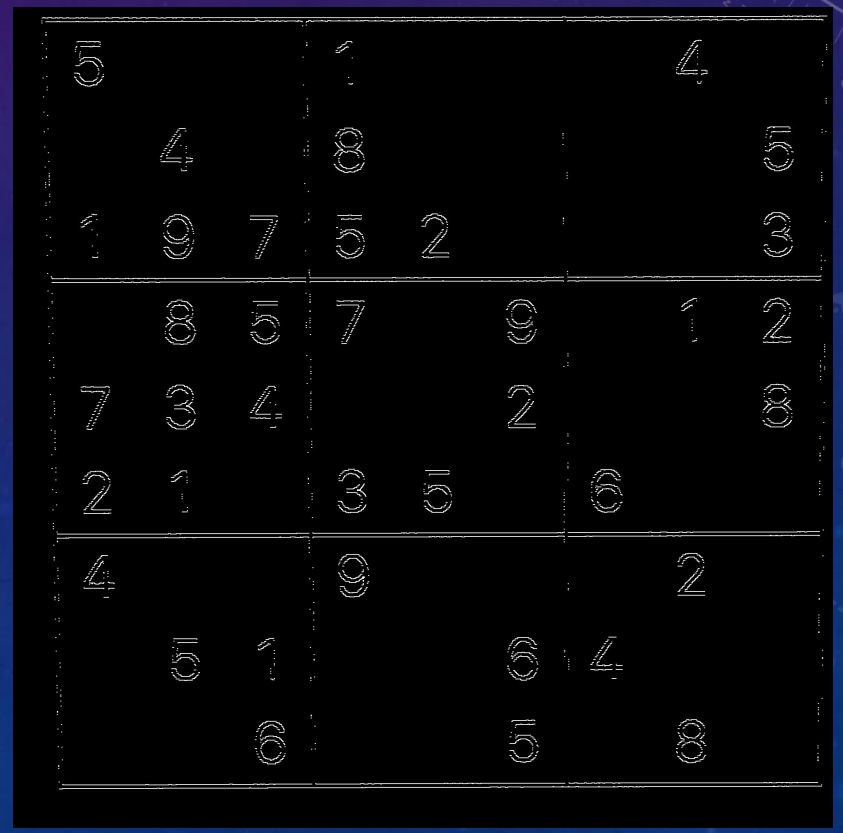
邊緣偵測

```python
#利用sobel偵測 vertical 與 horzation 的邊框
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobelX = np.uint8(np.absolute(sobelX))
cv2.imwrite("sobelX.jpg", sobelX)

sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1)
sobelY = np.uint8(np.absolute(sobelY))
cv2.imwrite("sobelY.jpg", sobelY)
```

Sobel X

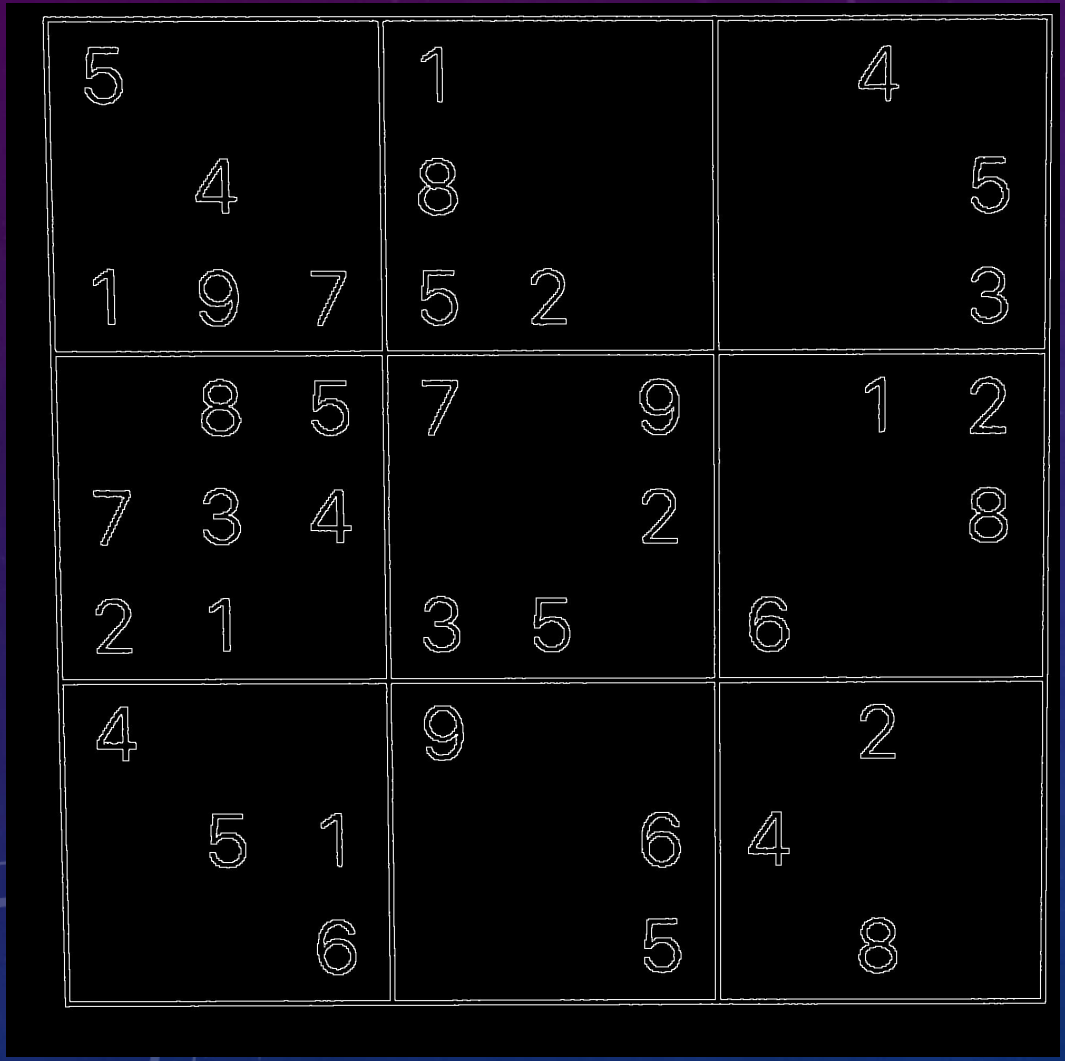


Sobel Y



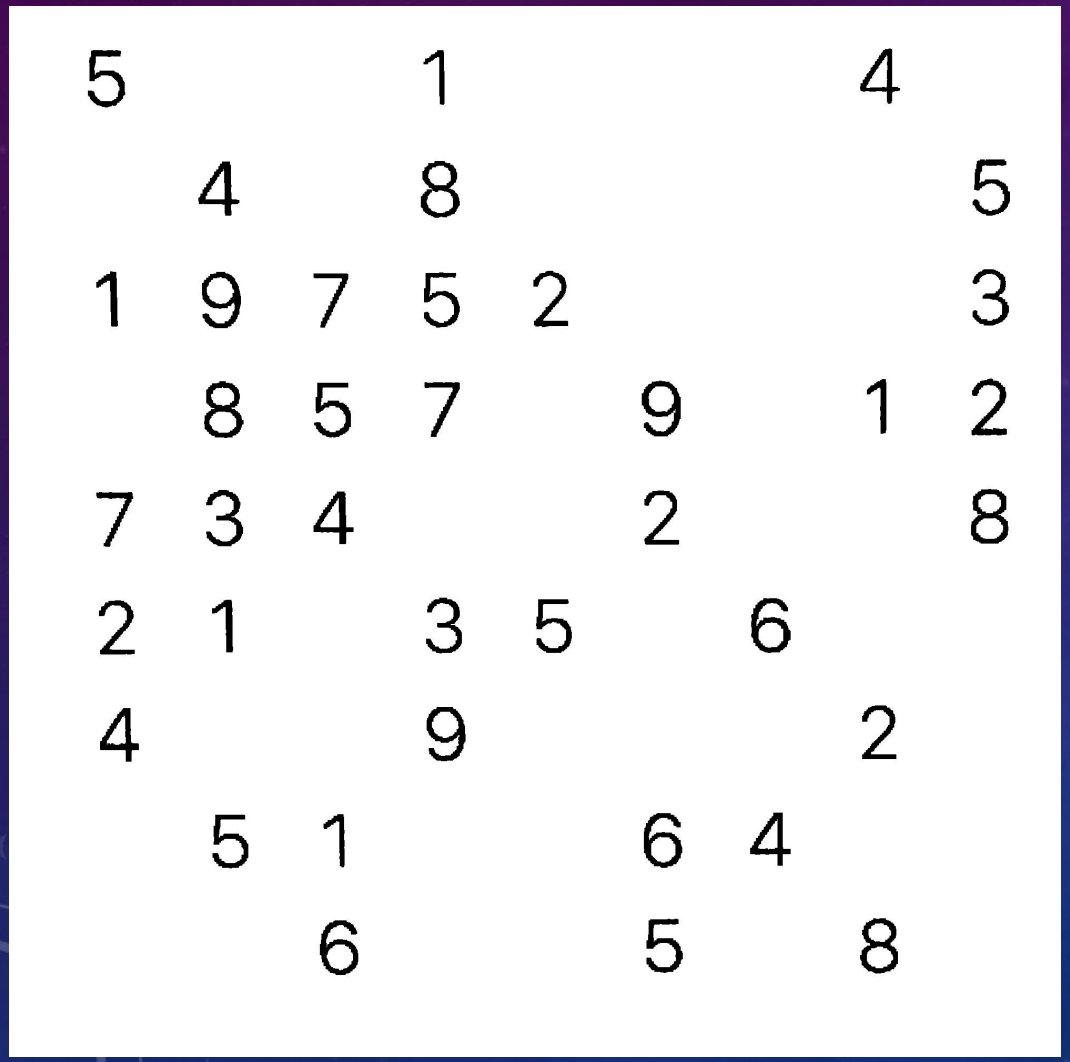
邊緣偵測-結合Sobel X 與 Sobel Y

Sobel XY



由於圖片中的邊框主要是直線與橫線，因此結合偵測直線與橫線的Sobel X 和 Sobel Y 來進行line detector

擦除邊框



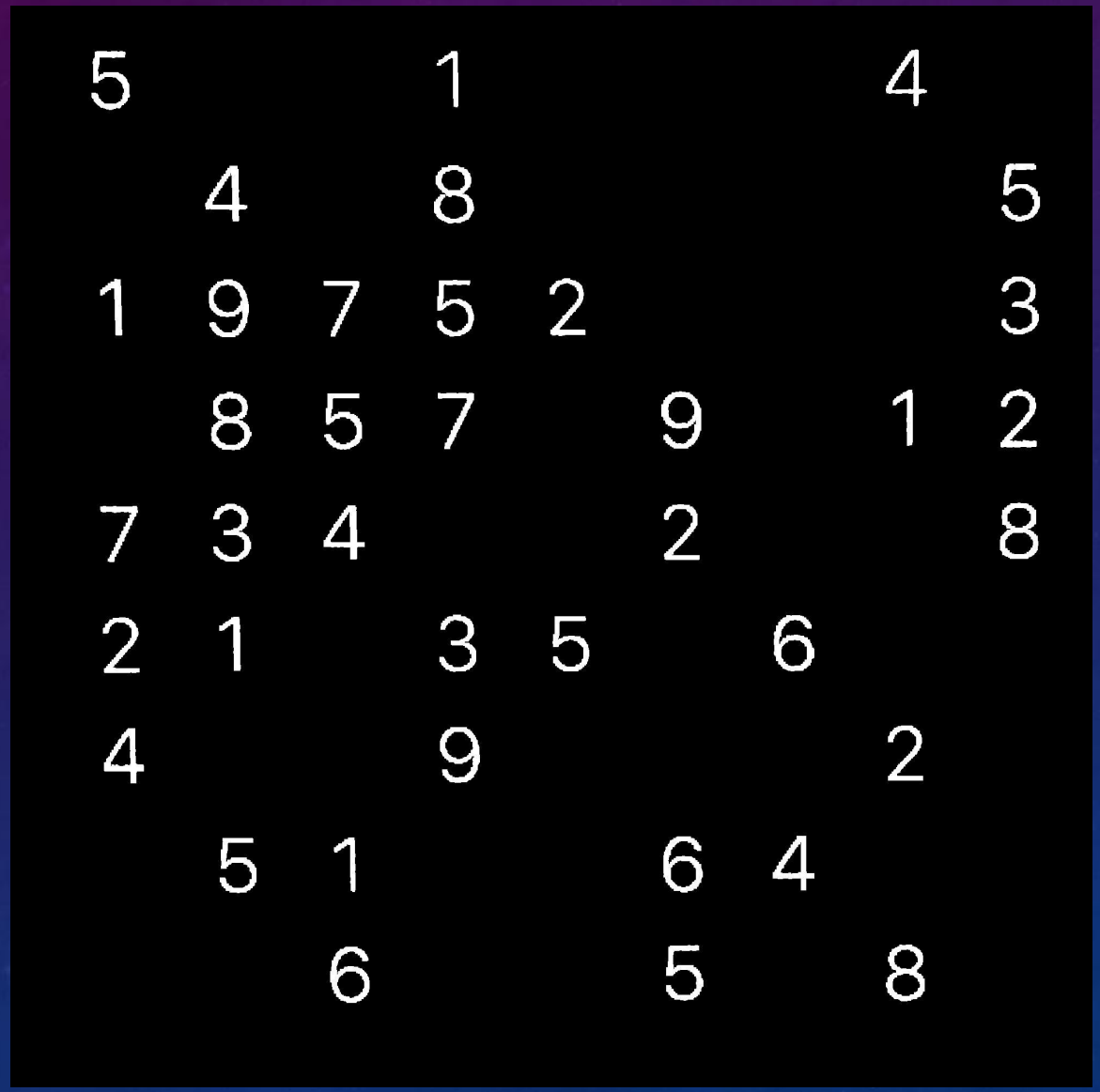

```python
#列出邊框大約的角度
angle_list = [1,2,3,4,5,6,7,8,9,10,85,86,87
              ,88,89,90,91,92,93,94,95]
for angle in angle_list:
    #使用Hough直線變換檢測直線
    lines = cv2.HoughLinesP(sobelXY,1,(np.pi/180)*angle,
                            200,minLineLength=150,
                            maxLineGap=40)
    #在原圖上用白色繪製檢測到的直線以擦除邊框
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(img, (x1, y1), (x2, y2), 255, 30)
```

利用Hough transform line detector 在計算出的邊緣圖片中計算直線橫線的位置，並在得出座標後在圖片中對應位置畫上白色達到擦除的效果

影像分析

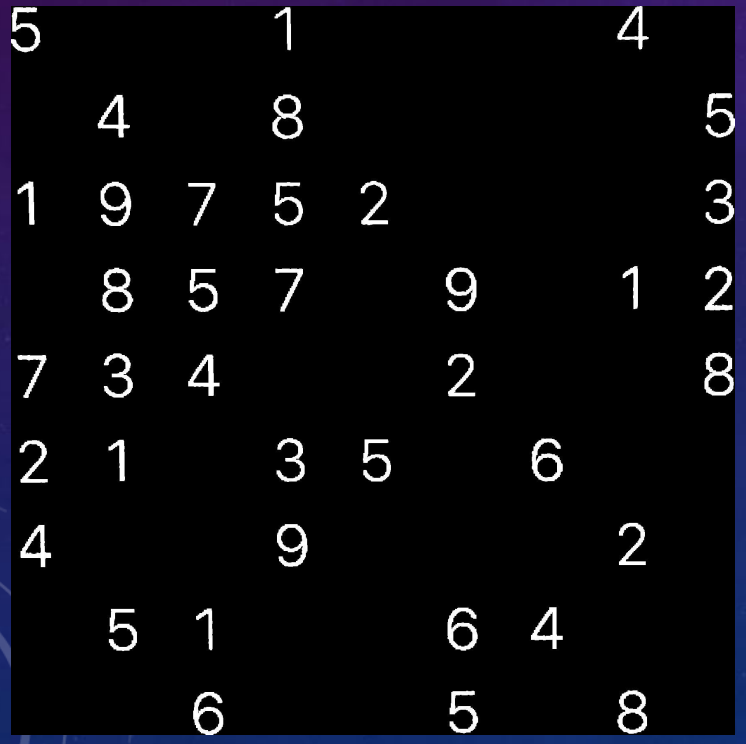
取負

將圖片取負方便計算邊界



切除外圍多餘部分

```
#將外圍的部分去除
hori_vals = np.sum(img, axis=1) #得到橫軸和的陣列用以判斷是否為邊界
hori_points = extractPeek(hori_vals) #得到行座標
img = img.crop((0, hori_points[0][0], img.width, hori_points[-1][1]))
vertical_vals = np.sum(img, axis=0)
vertical_points = extractPeek(vertical_vals) #得到列座標
img = img.crop((vertical_points[0][0], 0, vertical_points[-1][1], img.height))
```



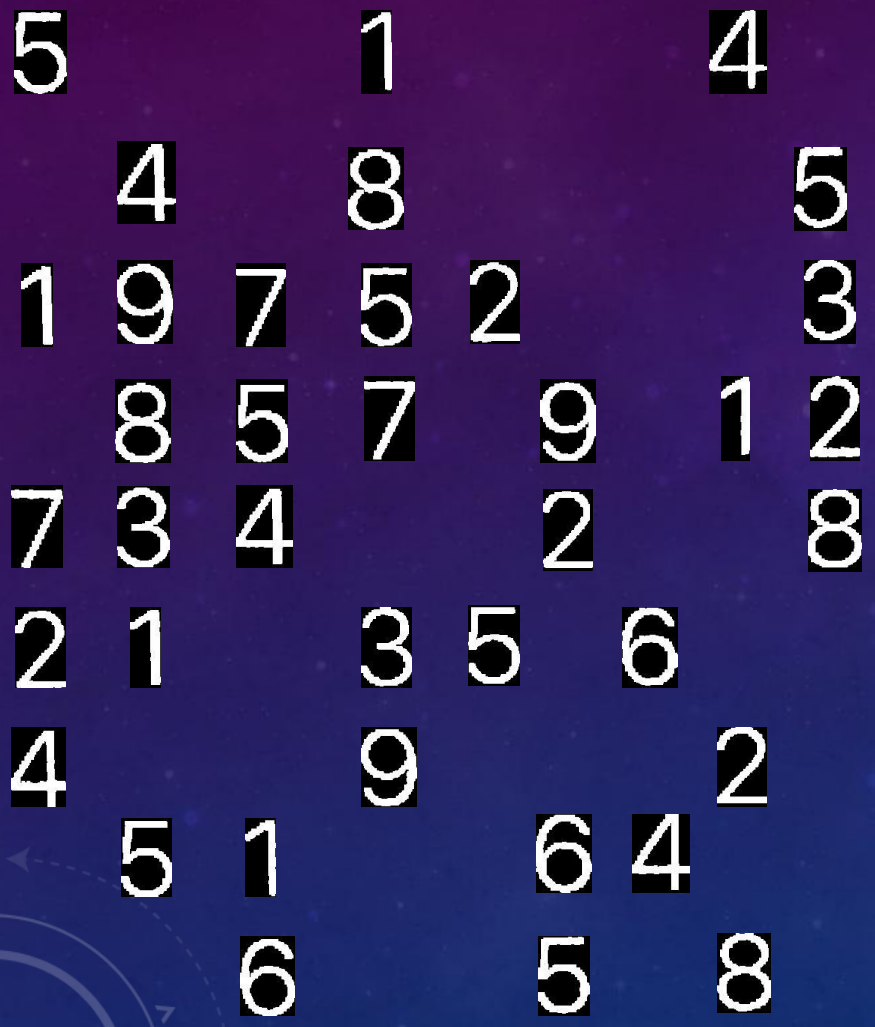
由於圖片中白色部分等於255，黑色部分為零分別計算圖Col、Row的合，當出現不為0的部分便可初步估計其為一條邊界

列切割

用相同的方式計算出橫向的邊界並切割

```python
#行掃描
hori_vals = np.sum(img, axis=1) #得到橫軸和的陣列用以判斷是否為邊界
hori_points = extractPeek(hori_vals) #得到行座標
#進行行切割
RowImg_list = []
for hori_point in hori_points:
    RowImg = img.crop((0, hori_point[0], img.width, hori_point[1])) #提取橫切割區域
    #儲存行圖片並記錄y座標
    RowImg_list.append([RowImg, hori_point[0], hori_point[1]])
    max_height = max(max_height, RowImg.height)
```

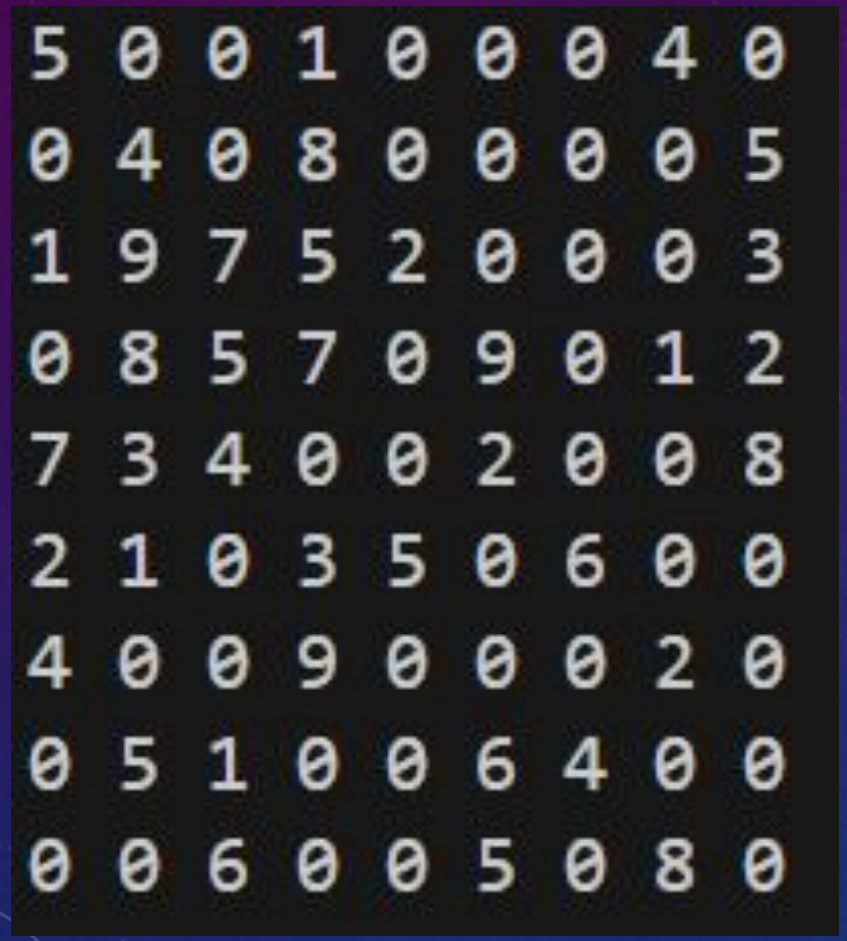
行切割



用相同的方式計算出豎向的邊界並切割

```
#進行列切割
NumImg_list = []
for RowImg,y0,y1 in RowImg_list:
    vec_vals = np.sum(RowImg,axis=0) #得到縱軸和之陣列用以判斷邊界
    vec_points = extractPeek(vec_vals, min_rect=20)

    nums = 0

    for vec_point in vec_points:
        IndividualImg = RowImg.crop((vec_point[0], 0, vec_point[1], RowImg.height))#依左上角以及右下角座標提取

        IndividualImg.save("individual_line" + str(line) + "_" + str(nums) + ".jpg")

        #再進行一次行切割得到最適合大小
        hori_vals = np.sum(IndividualImg, axis=1) #得到橫軸和的陣列用以判斷是否為邊界
        start_point, end_point = SignalExtract(hori_vals) #得到行座標
        IndividualImg = IndividualImg.crop((0, start_point, IndividualImg.width, end_point))
```

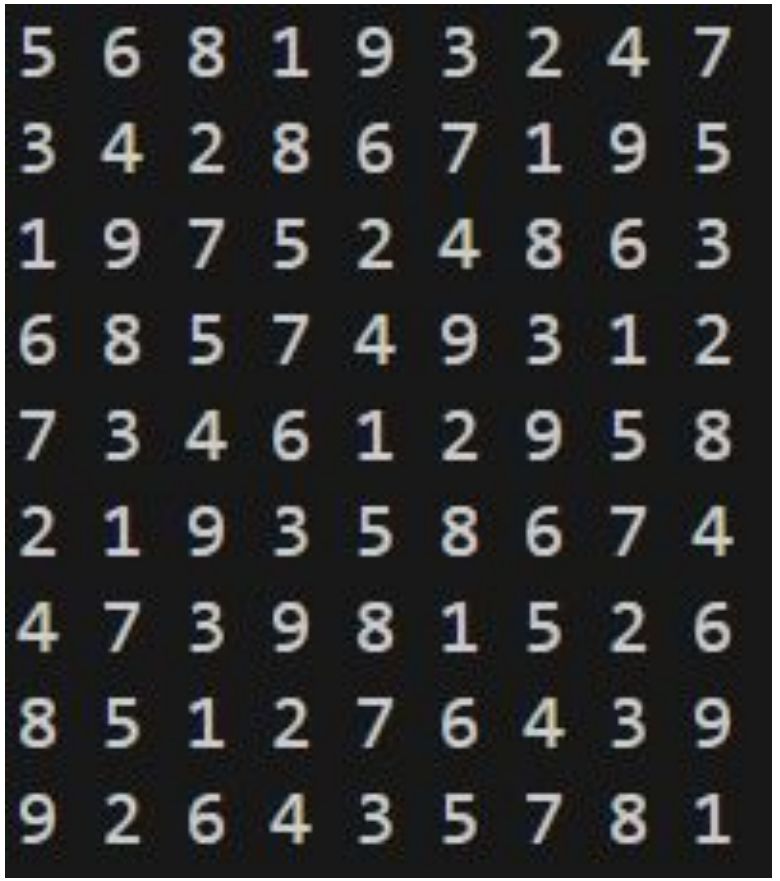
數字分析



利用第三方庫 ddddocr 分析切割出的圖片，並根據其在圖片中的對應位置計算出該圖片應該位於sudoku的哪裡，並建立sudoku的array

```
for pic in Num_list:
    col =  pic[1]
    row = pic[2]

    reImage = pic[0].resize((28,28))
    textI = ocr.classification(reImage)
    nine_grids[col][row] = textI
    if textI < '0' or textI > '9':
        return []
```

計算答案


5 6 8 1 9 3 2 4 7
3 4 2 8 6 7 1 9 5
1 9 7 5 2 4 8 6 3
6 8 5 7 4 9 3 1 2
7 3 4 6 1 2 9 5 8
2 1 9 3 5 8 6 7 4
4 7 3 9 8 1 5 2 6
8 5 1 2 7 6 4 3 9
9 2 6 4 3 5 7 8 1