Mathematical Image Processing Report
(License plate recognition with deep learning)

Chun-sheng Chen January 11, 2021
Source code: `https://github.com/JasonCheeeeen`

1 Abstract

License plate recognition is ubiquitous in parking lots, so i want to implement it by myself with methods of image processing. In order to increase its recogntion's speed, i used deep learning(CNN) to train the model of number and alphabet. Finally, using thread to recognize. All of these were implemented by python.

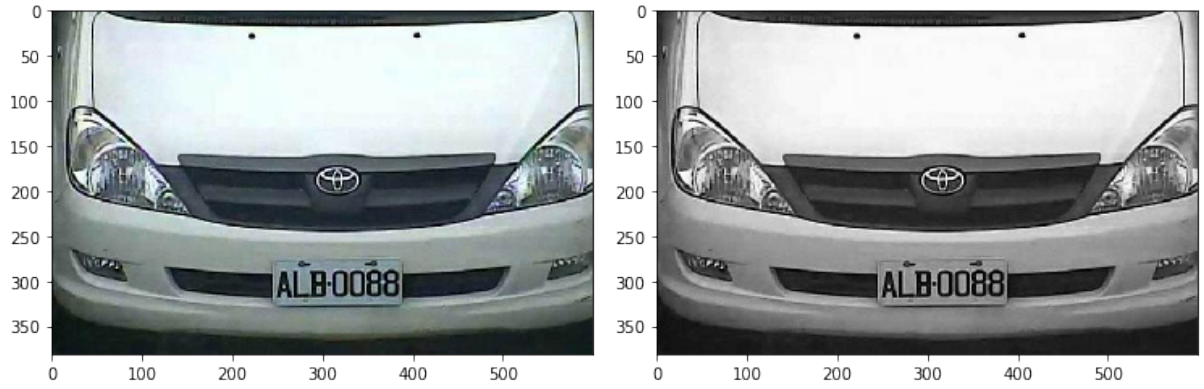
2 Algorithm

Algorithm 1: Get the license plate

1: **Input:** Original image captioned by camera (Figure. 2a)
2: Image grayscale standardization (Figure. 2b)
3: Using Canny to find the edges of image (Figure. 3a)
4: Using GaussianBlur remove some Impuritie (Figure. 3b)
5: Image Binarization and using open and close to find the license plate (Figure. 4a)
6: Get the coordinate of license plate and get the license plate (Algorithm. 2, Figure. 4b)
7: Cut the white space of above and below of image (Algorithm. 3, Figure. 5b)
8: Get the line of word in license plate (Algorithm. 4, Figure. 6a)
9: Cut the words by the line got from algorithm 1-8 (Algorithm. 5, Figure. 6b)
10: **return** Cutted number and alphabet

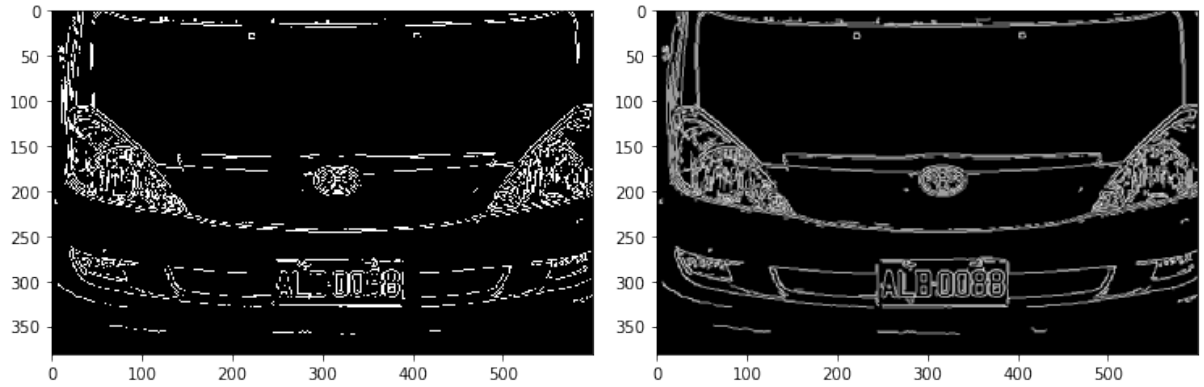
```
'''the kernal to dilate and erode'''
element1 = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 7))
element2 = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 3))
element3 = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 5))
element4 = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 10))
element5 = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

'''use open and close'''
img = cv2.dilate(img, element1, iterations = 2)
img = cv2.erode(img, element1, iterations = 2)
img = cv2.erode(img, element2, iterations = 10)
img = cv2.erode(img, element5, iterations = 5)
img = cv2.dilate(img, element5, iterations = 5)
img = cv2.dilate(img, element2, iterations = 10)
img = cv2.erode(img, element3, iterations = 10)
img = cv2.dilate(img, element3, iterations = 10)
img = cv2.erode(img, element4, iterations = 5)
img = cv2.dilate(img, element4, iterations = 5)
```

Figure 1: Parameters to erode or dilate



(a) Original image (b) After algoruthm 1-2

Figure 2: Origin and algorithm 1-2



(a) After algorithm 1-3 (b) After algorithm 1-4

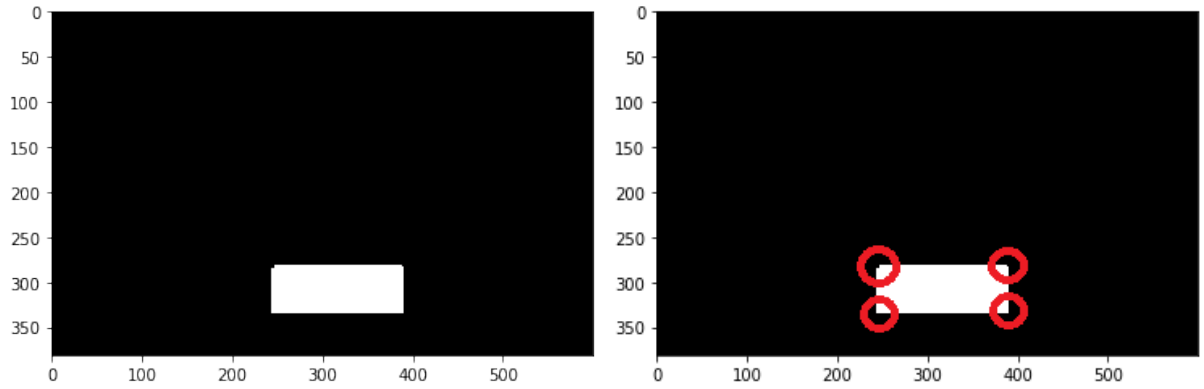
Figure 3: Algorithm 1-3 & 1-4

In Algorithm 1-3, if we want to use canny method to find the edge of image, it need parameters about threshold. Therefore, i take low threshold 100 and high thresnold 200 to implement. Beside of this, we also need to remove some impuritic in this image, so i used GaussianBlur to remove, and its kernal size is equal to 3. Moreover, there have many ways to remove impuritic, like median filter or mean filter, but the result of both were worse than GaussianBlur, because both of them destroyed the image in some section.

Algorithm 2: Algorithm 1 - 6

input : Image after Algorithm 1 - 5
output: coordinate of license plate

```
site_i ← [ ]
site_j ← [ ]
for i ← 0 to len(image) − 1 do
    for j ← 0 to len(image[0]) − 1 do
        if image[i][j][0]! = 0 then
            site_i.append(i)
            site_j.append(j)
        end
    end
end
widthleft ← site_i[0]
widthright ← site_i[-1]
heightup ← site_j[0]
heightdown ← site_j[-1]
return widthleft, widthright, heightup, heightdown
```



(a) After algorithm 1-5 (b) Get the coordinate

Figure 4: Algorithm 1-5 & 1-6

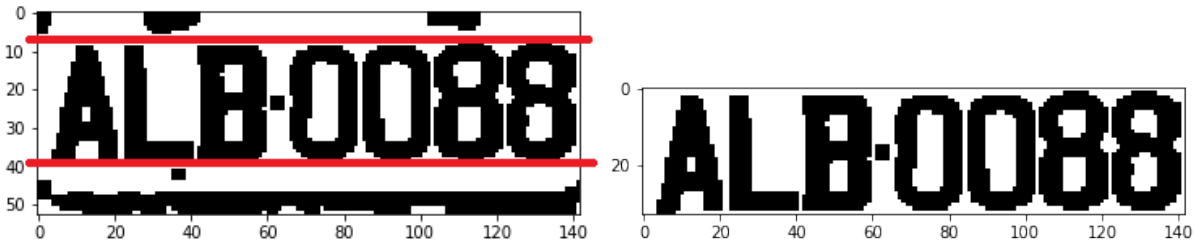
In order to use open and close method, we need to set the parameters of erode and dilate (Figure. 1). After using open and close, we can get the section of white space, so we can use (Algorithm. 2) to find the coordinate of white sapce and then cut this section.

Algorithm 3: Algorithm 1 - 7

input : Image after Algorithm 1 - 6 and Binarization

```
i_allzero ← [ ]
for i ← 0 to len(image) − 1 do
    count← 0
    for j ← 0 to len(image[0]) − 1 do
        if image[i][j] == 0 then
            count += 1
        end
    end
    i_allzero.append(count)
end
i_probable← [ ]
for i ← 0 to len(i_allzero) − 1 do
    if int(i_allzero[i]) < (1/10) ∗ len(image[0]) then
        i_probable.append(i)
    end
end
i_min ← [ ]
i_max ← [ ]
max_count ← [ ]
for i ← 0 to len(i_probable) − 1 do
    count = i_probable[i+1] - i_probable[i]
    if count > max_count then
        max_count = count
        i_min = i_probable[i]
        i_max = i_probable[i+1]
    end
end
image = image[i_min:i_max,:]
return image
```

In this algorithm, since there have two parts of white color in license plate, that is, above and low of the plate, we need to remove them. The way i used is to find the amount of width which pixel is zero for each height, after, finding the max difference between two adjacent index, then we can get the image which had benn cutted.



(a) After algorithm 1-6 (b) After algorithm 1-7

Figure 5: Algorithm 1-7

Algorithm 4: Algorithm 1 - 8

input : Image after Algorithm 1 - 7

```
j_allzero ← [ ]
for i ← 0 to len(image[0]) − 1 do
    count← 0
    for j ← 0 to len(image) − 1 do
        if image[i][j] == 0 then
            count += 1
        end
    end
    j_allzero.append(count)
end
j_probable← [ ]
for i ← 0 to len(j_allzero) − 1 do
    if int(j_allzero[i]) == len(image) then
        j_probable.append(i)
    end
end
j_final← [ ]
for i ← 0 to len(j_probable) − 1 do
    if j_probable[i + 1] − j_probable[i] > 1 then
        j_final.append(j_probable[i])
    end
end
return j_final
```

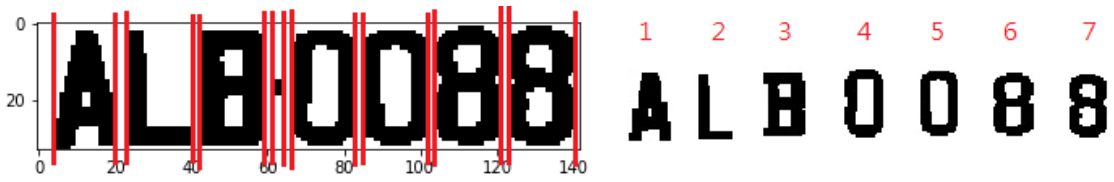
In this algorithm, since we wanted to get the single word of license plate, we got the probable straight line, then we could cut each word in this figure.

In our license plate, it contained three numbers, four alphabet and one dash. So we use index of dash be the standard and compute two kinds of length. Then, the longer length belongs to the alphabet, and numbers was the shorter length.

Algorithm 5: Algorithm 1 - 9

input : Image after Algorithm 1 - 8 and j_final

```
value ← [ ]
for i ← 0 to len(j_final) − 1 do
    value.append(j_final[i+1]-j_final[i])
end
dash ← min(value)
index ← 0
for i ← 0 to len(j_final) − 1 do
    if int(value[i]) == dash then
        dash_index = j_final[i:i+2]
        index = i
    end
end
length1 = j_final[index] - j_final[0]
length2 = j_final[-1] - j_final[index+1]
j_final← [ ]
if length2 > length1 then
    if len(j_final) == 9 then
        card_eng = card_bw[:j_final[0]:j_final[3]]
        card_dash = card_bw[:j_final[3]:j_final[4]]
        card_num = card_bw[:j_final[4]:]
    end
end
use index to cut each word
return numbers, alphabet
```



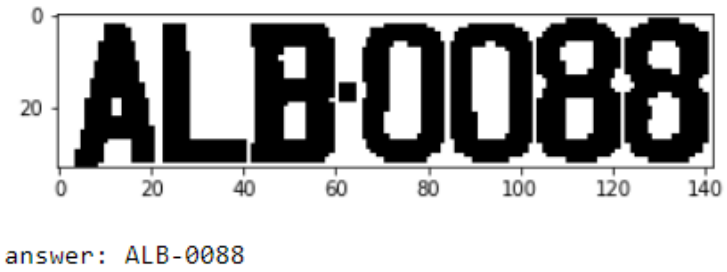
(a) After algorithm 1-8 (b) After algorithm 1-9

Figure 6: Algorithm 1-8 & 1-9

Algorithm 6: Word recognition by Convolution Neural Network

1: **Input:** Cutted numbers and alphabet (Figure. 6b)
2: Train CNN model of numbers and alphabet
3: Put input data into trainned model
4: **return** Identified numbers and alphabet

Result:



3 Conclusion

In this project, if we input a original picture to algoruthm 1, it needs approximately 5 seconds to recognize. But, if we input a picture which license plate was cutted or we have the proper coordinate of license plate, like parkint lot in department store. In other words, we could save time to track license plate, that is, we only need to cut words respectively, and put into algorithm 6 directly. Also, in this project, i used way of "thread" to increase speed. Finally, we could minimize the recognition time within 2 seconds.

4 Reference

(1) Images: Google

(2) Mnist: `https://www.tensorflow.org/datasets/catalog/mnist`

(3) A∼Z alphabet: `https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-f`