EE6222 Assignment1 option2

ZHANG ZHENGHANG

E-mail: ZZHANG062@e.ntu.edu.sg

1.TheThresholding Process:

Otus's method

The main solution of this method is to find a "K" which can maximize the between-class variance."K" is a pending value coming from Axis of the gray levels histogram.

By randomly assume the value of "K", all the pixels can be divided into two parts. It's easy to obtain the mean and the variance of these two parts.

So we calculate the variance between these two parts(σ_B^2). By trying different value of "K", we assume the "K" to obtain the maximum of σ_B^2 as the optical threshold.

```
def otus_threshold(image):
    n, m = image.shape
    total_pixel = n * m
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])

    # 总平均灰度
    u = 0.0
    for i in range(len(hist)):
        u = u + i * hist[i]
    u = u / total_pixel

    # 背景灰度级平均灰度
    u_0 = []
    u_0_mid = 0.0
    w_0 = []
    for j in range(len(hist)):
        back_pixel = np.sum(hist[:j+1])
        w_0.append(back_pixel / total_pixel)
        u_0_mid = u_0_mid + j * hist[j]
        if back_pixel == 0:
            u_0.append(0.0)
        else:
            u_0.append(u_0_mid / back_pixel)
    # print(u_0)

    # 类间方差
    g = []
    a = 0.0
    for k in range(len(hist)):
        if w_0[k] == 1:
            g.append(a)
        else:
            u_1 = (u - u_0[k] * w_0[k]) / (1 - w_0[k])
            g_mid = w_0[k] * (1.0 - w_0[k]) * (u_0[k] - u_1) * (u_0[k] - u_1)
            g.append(g_mid)

    # print(len(g), g)
    g_index = g.index(max(g))

    return g_index
```

2. The Thinning Algorithm:

My thinning code is based on the two-step thinning algorithm introduced in class. In my code, a matrix only consist of “0” and “1” will be processed by ”Zhang_suen_thinning”, “o” means this pixel belong to group ”background” while “1” means “object”.

STEP1:Traverse every pixels, if one pixel meets the following conditions, it will be set to ”0”.(Background)

(1) if its value is “1”.

(2) 2<=The number of Its non-zero neighbors<=6

(3) S=1. S is the the number of 0-1 transition in the ordered sequence of p2,p3,..,p8,p9.

(4) At least one of p2,p4,p6 is equal to “0”

(5) At least one of p2,p4,p6 is equal to “0”

STEP2:Traverse every pixels, if one pixel meets the following conditions, it will be set to ”0”.(Background)

(6) if its value is “1”.

(7) 2<=The number of Its non-zero neighbors<=6

(8) S=1. S is the the number of 0-1 transition in the ordered sequence of p2,p3,..,p8,p9.

(9) At least one of p2,p4,p8 is equal to “0”

(10) At least one of p2,p6,p8 is equal to “0”

The steps are repeated until there is no change to all pixels.

```python
def neighbours(x, y, image):
    '''Return 8-neighbours of point p1 of picture, in order'''
    i = image
    x1, y1, x_1, y_1 = x + 1, y - 1, x - 1, y + 1
    # print ((x,y))
    return [i[y1][x], i[y1][x1], i[y][x1], i[y_1][x1],   # P2,P3,P4,P5
            i[y_1][x], i[y_1][x_1], i[y][x_1], i[y1][x_1]]  # P6,P7,P8,P9


def transitions(neighbours):
    n = neighbours + neighbours[0:1]  # P2, ... P9, P2
    return sum((n1, n2) == (0, 1) for n1, n2 in zip(n, n[1:]))


def thining(image):
    changing1 = changing2 = [(-1, -1)]
    while changing1 or changing2:
        # Step 1
        changing1 = []
        for y in range(1, len(image) - 1):
            for x in range(1, len(image[0]) - 1):
                P2, P3, P4, P5, P6, P7, P8, P9 = n = neighbours(x, y, image)
                if (image[y][x] == 1 and  # (Condition 0)
                        P4 * P6 * P8 == 0 and  # Condition 4
                        P2 * P4 * P6 == 0 and  # Condition 3
                        transitions(n) == 1 and  # Condition 2
                        2 <= sum(n) <= 6):  # Condition 1
                    changing1.append((x, y))
        for x, y in changing1: image[y][x] = 0
        # Step 2
        changing2 = []
        for y in range(1, len(image) - 1):
            for x in range(1, len(image[0]) - 1):
                P2, P3, P4, P5, P6, P7, P8, P9 = n = neighbours(x, y, image)
                if (image[y][x] == 1 and  # (Condition 0)
                        P2 * P6 * P8 == 0 and  # Condition 4
                        P2 * P4 * P8 == 0 and  # Condition 3
                        transitions(n) == 1 and  # Condition 2
                        2 <= sum(n) <= 6):  # Condition 1
                    changing2.append((x, y))
        for x, y in changing2: image[y][x] = 0
        # print changing1

    #image = image.astype(np.uint8) * 255
    return image
```

NOTE:It is necessary to let the processed matrix multiple with 255, or the matrix is still (0,1) binary, which will lead to a all-black image.

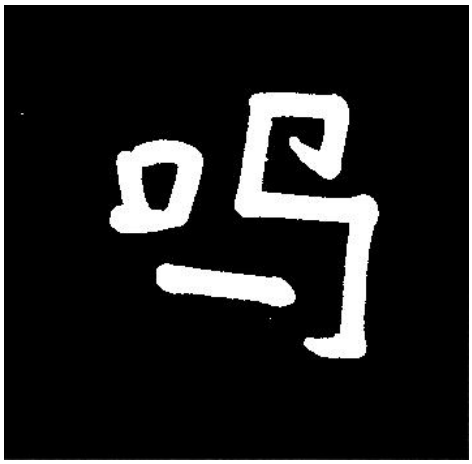
3. The intermediate and final results:

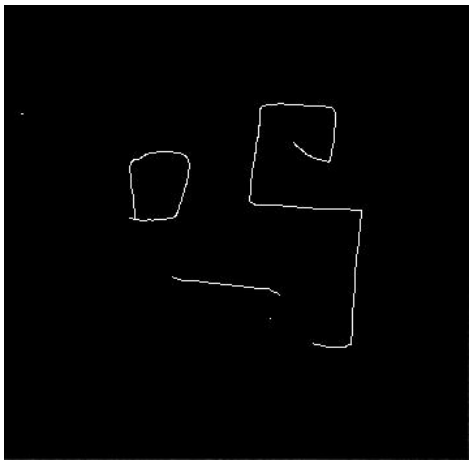
Img1:

After the thresholding process, the optical “K” is calculated to be set to ”188”.

Img2:

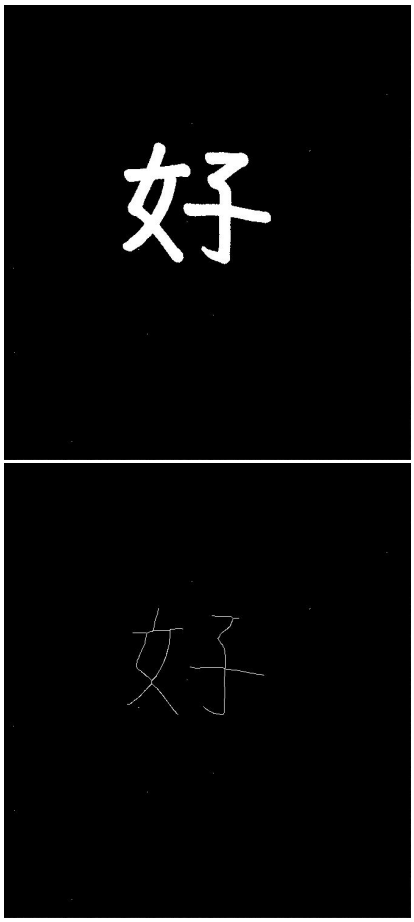
After the thresholding process, the optical “K” is calculated to be set to ”120”.





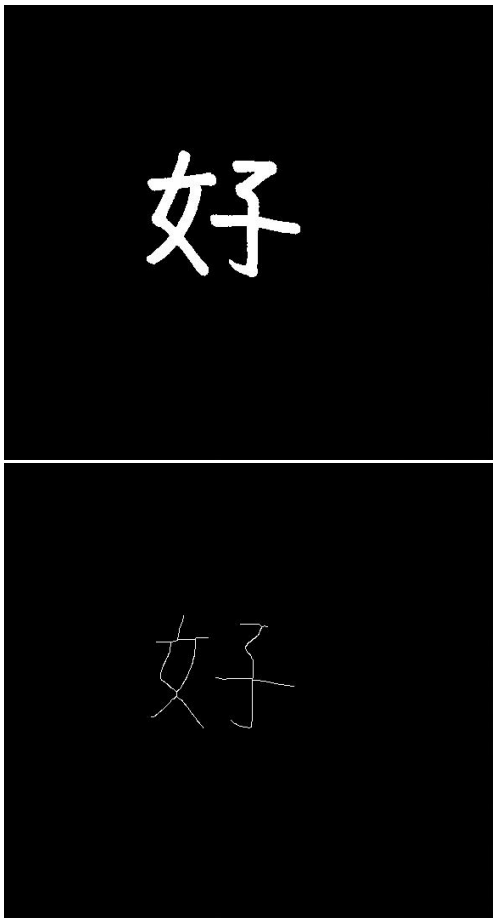
Img3:

After the thresholding process, the optical “K” is calculated to be set to ”119”.



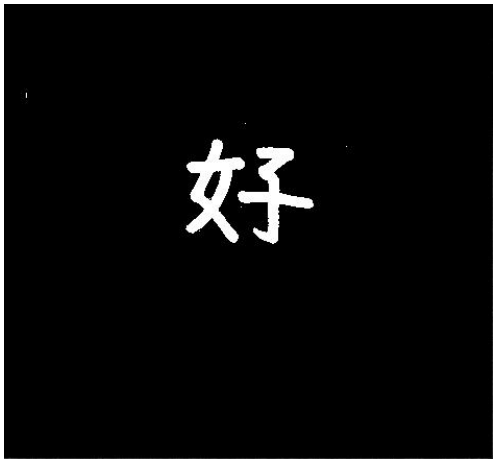
Img4:

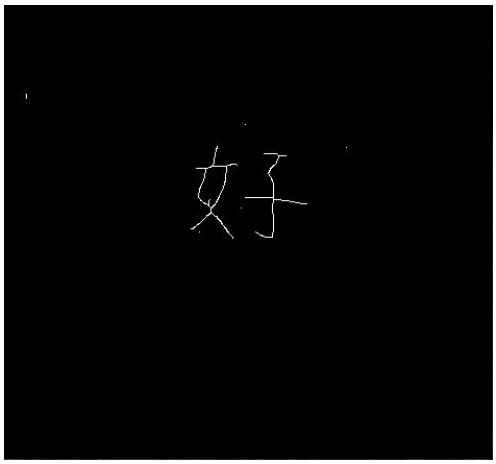
After the thresholding process, the optical “K” is calculated to be set to ”208”.



Img5:

After the thresholding process, the optical “K” is calculated to be set to ”114”.





Comment:

The quality of result is ideal. The thresholding process is very useful, which will reduce the work to find the optical value by different experimental results.

4. Python code:

1.read the pending image and translate this image into matrix. The element of this matrix means the gray level of every pixel.(0-255)

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.image
from PIL import Image
np.set_printoptions(threshold=np.inf)
```

```
im = Image.open("C:\\Users\\86198\\Desktop\\MV ASS1\\img6.bmp")
```

```
im.show()
```

```
im = im.convert("L")
```

```
im.show()
```

```
matrix = np.asarray(im)
```

```
amatrix = matrix.copy()
```

2.Use the optical threshold to translate the above matrix into a (0,1) binary matrix.

```
threshold=otus_threshold(amatrix)
```

```
for i in range(0,amatrix.shape[0]-1):
    for j in range(0,amatrix.shape[1]-1):
        if  matrix[i][j]>=threshold:
            amatrix[i][j]=0
        else:
            amatrix[i][j]=1
```

3. show the immediate and final result.

```
Namatrix = amatrix.astype(np.uint8) * 255
new_im = Image.fromarray(Namatrix)
new_im.show()
```

```
out = thining(amatrix)
```

```
out = out.astype(np.uint8) * 255
```

```
new_im = Image.fromarray(out)
```

```
new_im.show()
```