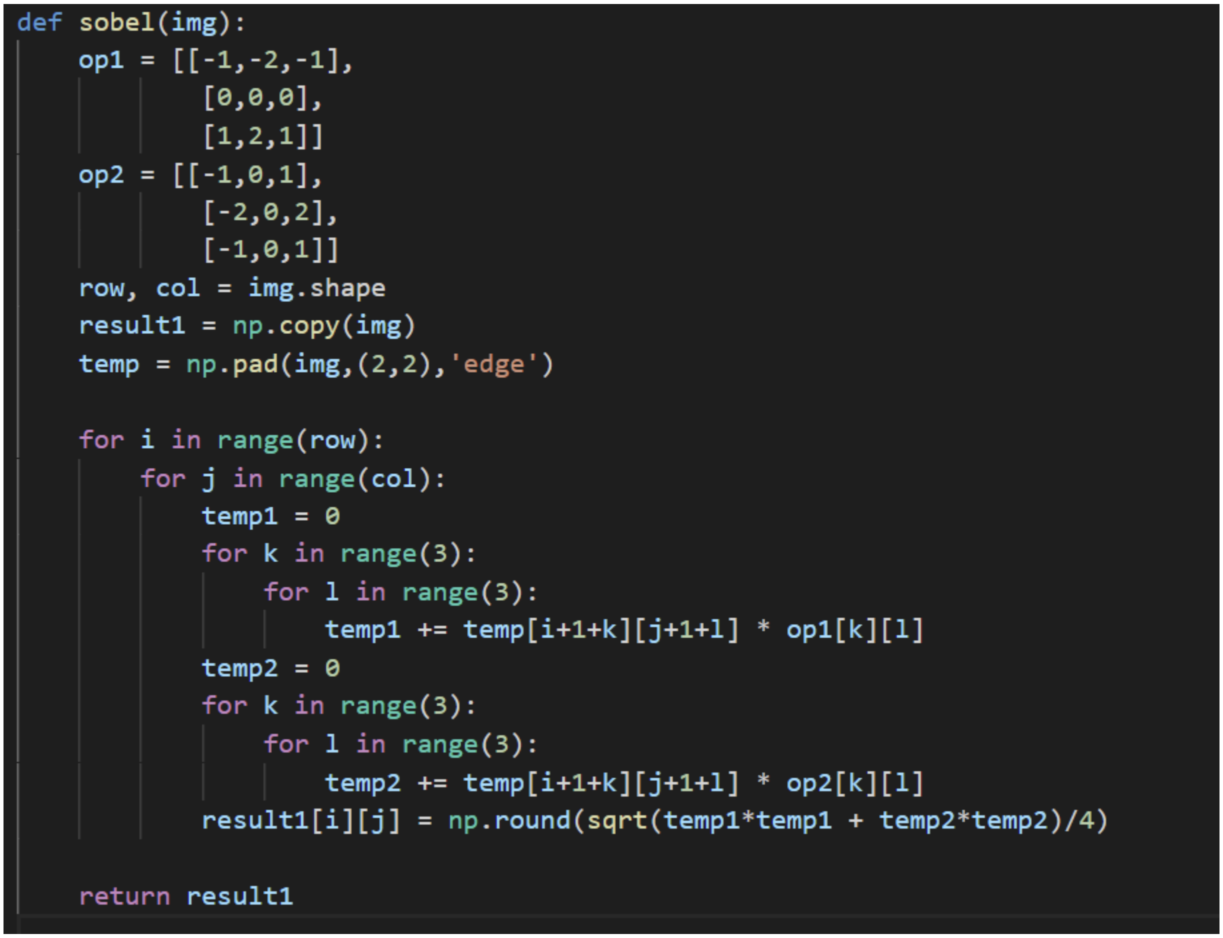
HOMEWORK ASSIGNMENT 2

R10922A16 蔡家豪

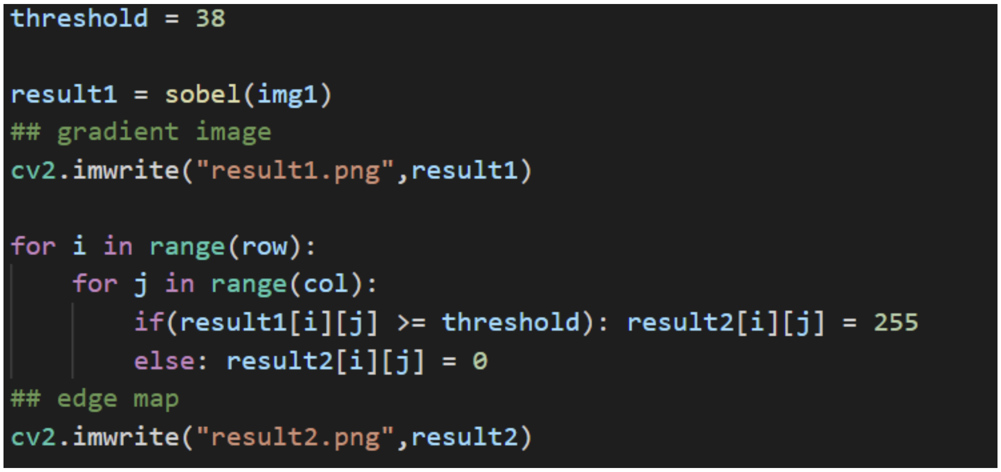
Problem 1: EDGE DETECTION

(a)Sobel edge detection:

Method:



```python
def sobel(img):
    op1 = [[-1,-2,-1],
            [0,0,0],
            [1,2,1]]
    op2 = [[-1,0,1],
            [-2,0,2],
            [-1,0,1]]
    row, col = img.shape
    result1 = np.copy(img)
    temp = np.pad(img,(2,2),'edge')

    for i in range(row):
        for j in range(col):
            temp1 = 0
            for k in range(3):
                for l in range(3):
                    temp1 += temp[i+1+k][j+1+l] * op1[k][l]
            temp2 = 0
            for k in range(3):
                for l in range(3):
                    temp2 += temp[i+1+k][j+1+l] * op2[k][l]
            result1[i][j] = np.round(sqrt(temp1*temp1 + temp2*temp2)/4)

    return result1
```



```
threshold = 38

result1 = sobel(img1)
## gradient image
cv2.imwrite("result1.png",result1)

for i in range(row):
    for j in range(col):
        if(result1[i][j] >= threshold): result2[i][j] = 255
        else: result2[i][j] = 0
## edge map
cv2.imwrite("result2.png",result2)
```

Use threshold = 38

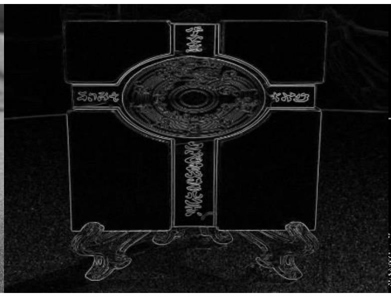
If you use lower threshold, the noise will be more than higher one.

The higher threshold will lose more detail, I think 38 is one of the most balance threshold.

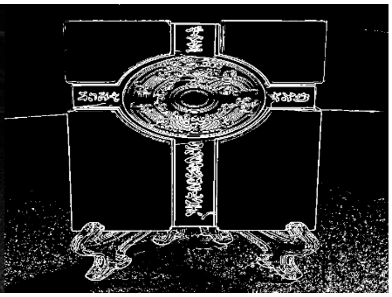
Images:

Sample1

result1(gradient image)



result2(edge map)

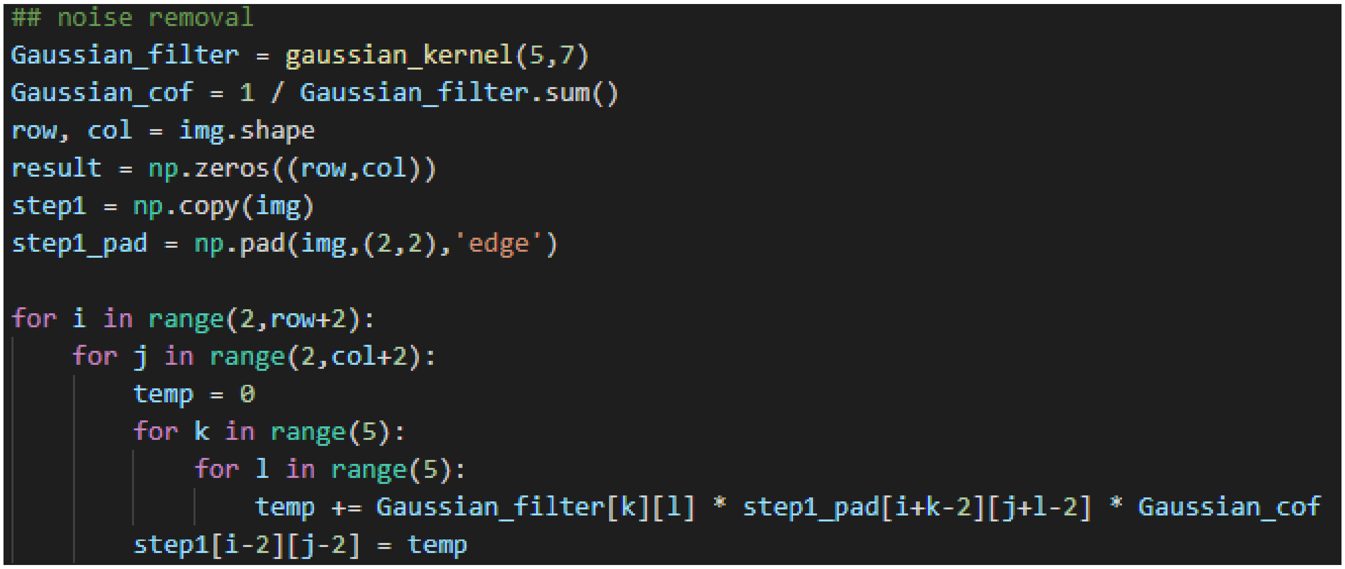


(b)Canny edge detection:

Method:

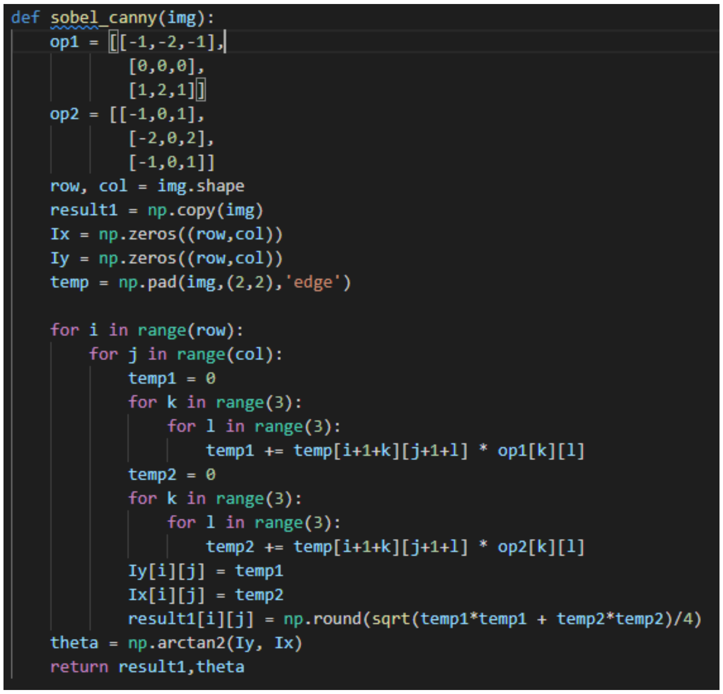
1.noise removal:

Use gaussian kernel to remove noise, and need to use higher sigma or your noise will be too much.



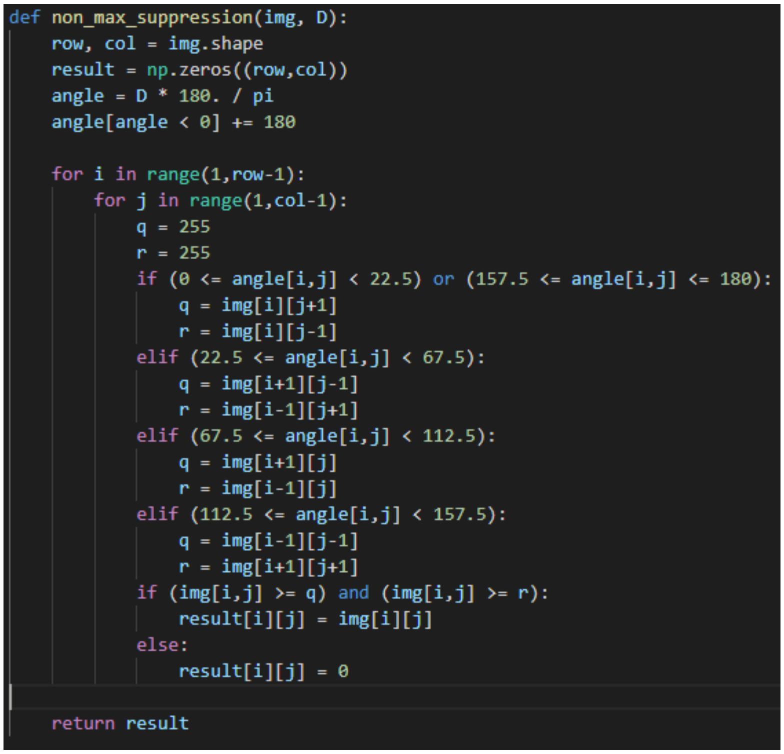
```python
## noise removal
Gaussian_filter = gaussian_kernel(5,7)
Gaussian_cof = 1 / Gaussian_filter.sum()
row, col = img.shape
result = np.zeros((row,col))
step1 = np.copy(img)
step1_pad = np.pad(img,(2,2),'edge')

for i in range(2,row+2):
    for j in range(2,col+2):
        temp = 0
        for k in range(5):
            for l in range(5):
                temp += Gaussian_filter[k][l] * step1_pad[i+k-2][j+l-2] * Gaussian_cof
        step1[i-2][j-2] = temp
```

2.use sobel to calculate gradient



```python
def sobel_canny(img):
    op1 = [[-1,-2,-1],
            [0,0,0],
            [1,2,1]]
    op2 = [[-1,0,1],
            [-2,0,2],
            [-1,0,1]]
    row, col = img.shape
    result1 = np.copy(img)
    Ix = np.zeros((row,col))
    Iy = np.zeros((row,col))
    temp = np.pad(img,(2,2),'edge')

    for i in range(row):
        for j in range(col):
            temp1 = 0
            for k in range(3):
                for l in range(3):
                    temp1 += temp[i+1+k][j+1+l] * op1[k][l]
            temp2 = 0
            for k in range(3):
                for l in range(3):
                    temp2 += temp[i+1+k][j+1+l] * op2[k][l]
            Iy[i][j] = temp1
            Ix[i][j] = temp2
            result1[i][j] = np.round(sqrt(temp1*temp1 + temp2*temp2)/4)
    theta = np.arctan2(Iy, Ix)
    return result1,theta
```

3. non_max_suppression



```
def non_max_suppression(img, D):
    row, col = img.shape
    result = np.zeros((row,col))
    angle = D * 180. / pi
    angle[angle < 0] += 180

    for i in range(1,row-1):
        for j in range(1,col-1):
            q = 255
            r = 255
            if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                q = img[i][j+1]
                r = img[i][j-1]
            elif (22.5 <= angle[i,j] < 67.5):
                q = img[i+1][j-1]
                r = img[i-1][j+1]
            elif (67.5 <= angle[i,j] < 112.5):
                q = img[i+1][j]
                r = img[i-1][j]
            elif (112.5 <= angle[i,j] < 157.5):
                q = img[i-1][j-1]
                r = img[i+1][j+1]
            if (img[i,j] >= q) and (img[i,j] >= r):
                result[i][j] = img[i][j]
            else:
                result[i][j] = 0

    return result
```

4. double_thresholiding

I think this part is most important part.

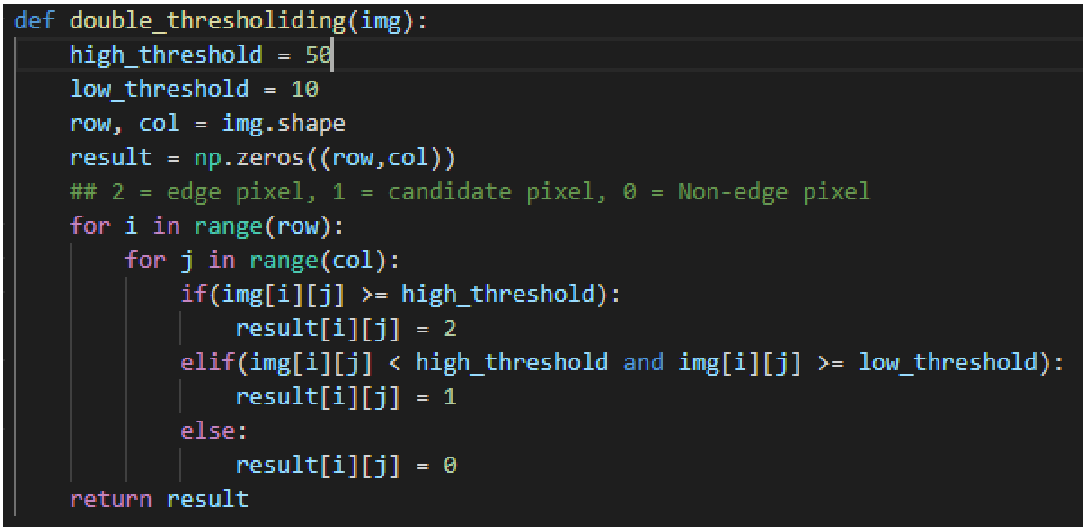
How to choose high and low threshold combo is a big question.

The lower low_threshold will keep more detail but the noises will become.

The higher low_threshold will lose a big part of information.

And actually high_threshold is not a big issue, just pick one does not lose too much detail.

I use 50 & 10 as my combo.

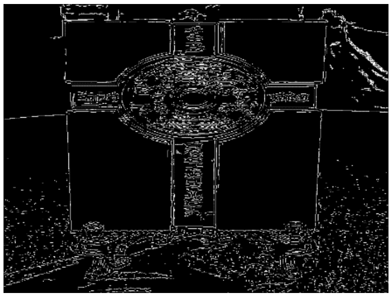


```
def double_thresholiding(img):
    high_threshold = 50
    low_threshold = 10
    row, col = img.shape
    result = np.zeros((row,col))
    ## 2 = edge pixel, 1 = candidate pixel, 0 = Non-edge pixel
    for i in range(row):
        for j in range(col):
            if(img[i][j] >= high_threshold):
                result[i][j] = 2
            elif(img[i][j] < high_threshold and img[i][j] >= low_threshold):
                result[i][j] = 1
            else:
                result[i][j] = 0
    return result
```

Image:

Sample1

result3



(c)Laplacian of Gaussian:

Compare to result 2 and result3

Result 4 lost some line and have more noise compare to result2

But compare to result3, result4 keep more information we focus on. (maybe the reason result4 is better than result3 is I screw up the result3 algorithm lol.)

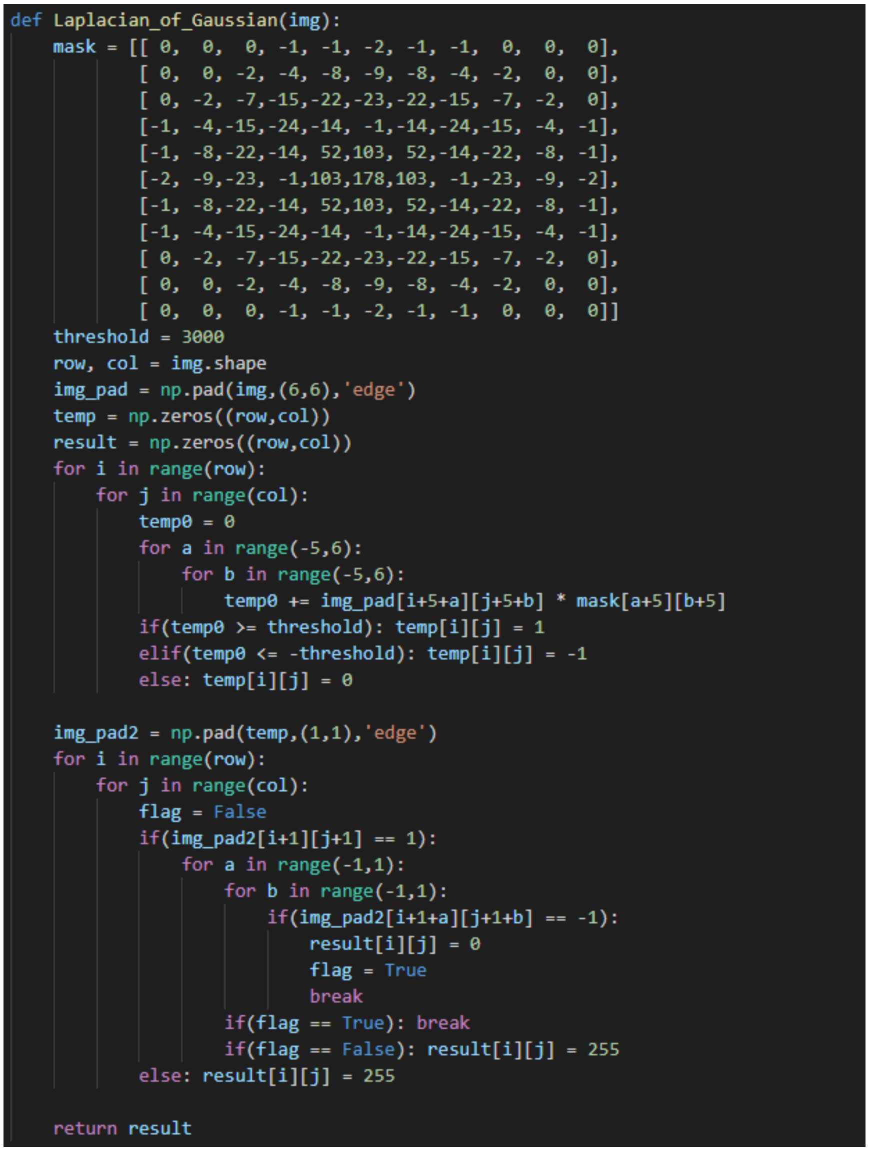
Method:

I use the mask I use in Computer vision course.

And the threshold I use is 3000

The higher threshold can make noise less, but will lose some lines.

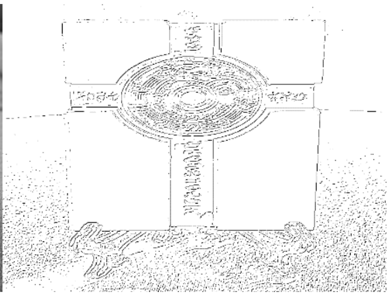
The lower threshold will have more clear lines, but will keep more noise in image.



```python
def Laplacian_of_Gaussian(img):
    mask = [[ 0,  0,  0, -1, -1, -2, -1, -1,  0,  0,  0],
            [ 0,  0, -2, -4, -8, -9, -8, -4, -2,  0,  0],
            [ 0, -2, -7,-15,-22,-23,-22,-15, -7, -2,  0],
            [-1, -4,-15,-24,-14, -1,-14,-24,-15, -4, -1],
            [-1, -8,-22,-14, 52,103, 52,-14,-22, -8, -1],
            [-2, -9,-23, -1,103,178,103, -1,-23, -9, -2],
            [-1, -8,-22,-14, 52,103, 52,-14,-22, -8, -1],
            [-1, -4,-15,-24,-14, -1,-14,-24,-15, -4, -1],
            [ 0, -2, -7,-15,-22,-23,-22,-15, -7, -2,  0],
            [ 0,  0, -2, -4, -8, -9, -8, -4, -2,  0,  0],
            [ 0,  0,  0, -1, -1, -2, -1, -1,  0,  0,  0]]
    threshold = 3000
    row, col = img.shape
    img_pad = np.pad(img,(6,6),'edge')
    temp = np.zeros((row,col))
    result = np.zeros((row,col))
    for i in range(row):
        for j in range(col):
            temp0 = 0
            for a in range(-5,6):
                for b in range(-5,6):
                    temp0 += img_pad[i+5+a][j+5+b] * mask[a+5][b+5]
            if(temp0 >= threshold): temp[i][j] = 1
            elif(temp0 <= -threshold): temp[i][j] = -1
            else: temp[i][j] = 0

    img_pad2 = np.pad(temp,(1,1),'edge')
    for i in range(row):
        for j in range(col):
            flag = False
            if(img_pad2[i+1][j+1] == 1):
                for a in range(-1,1):
                    for b in range(-1,1):
                        if(img_pad2[i+1+a][j+1+b] == -1):
                            result[i][j] = 0
                            flag = True
                            break
                    if(flag == True): break
                    if(flag == False): result[i][j] = 255
            else: result[i][j] = 255

    return result
```

Image:

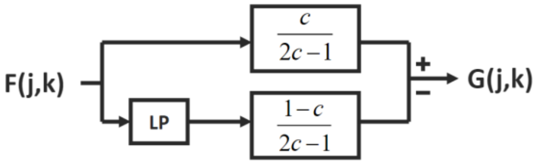
Sample1 result4



(d)Edge crispening:

Method:

Same as slide on course:



$$G(j,k)=\frac{c}{2c-1}F(j,k)-\frac{1-c}{2c-1}F_L(j,k), \quad where \quad \frac{3}{5}\le c\le\frac{5}{6}$$

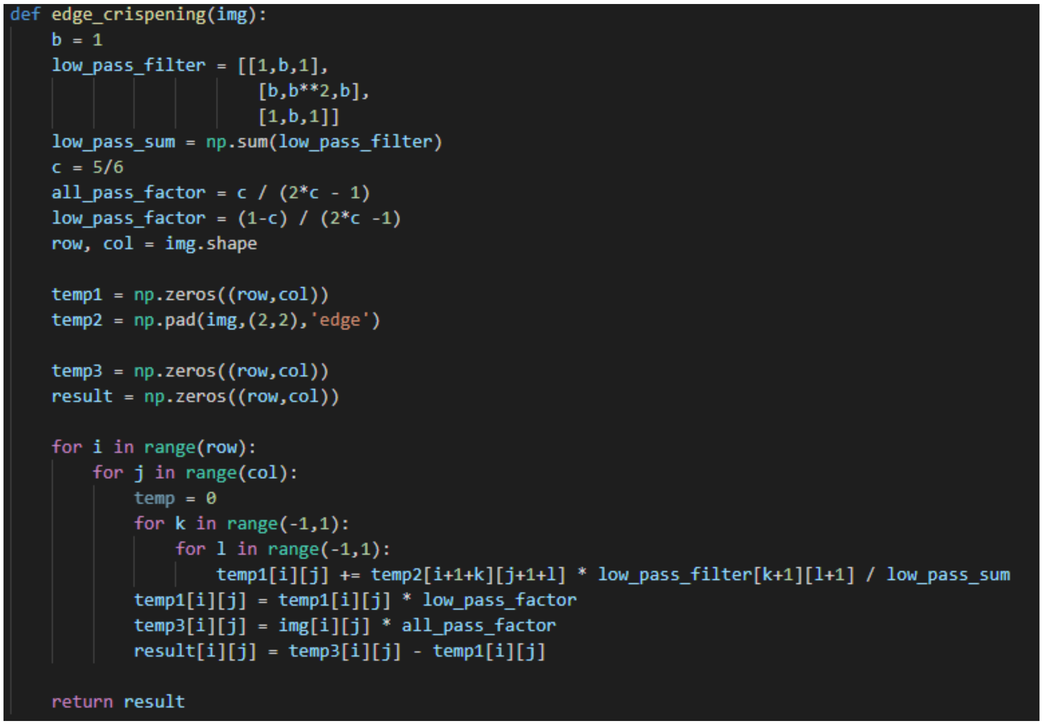
The 2 parameter is c and low pass filter

If you change c to 3/5 the whole image will become brighter, but I personally 3/5 is too bright, it lost so many details.

I prefer 5/6 to just adjust the original image a little bit.

And low pass filter If use b > 1, it will slightly change the bright but not obvious.

After edge crispening, the contrast will better than original.



```
def edge_crispening(img):
    b = 1
    low_pass_filter = [[1,b,1],
                       [b,b**2,b],
                       [1,b,1]]
    low_pass_sum = np.sum(low_pass_filter)
    c = 5/6
    all_pass_factor = c / (2*c - 1)
    low_pass_factor = (1-c) / (2*c -1)
    row, col = img.shape

    temp1 = np.zeros((row,col))
    temp2 = np.pad(img,(2,2),'edge')

    temp3 = np.zeros((row,col))
    result = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            temp = 0
            for k in range(-1,1):
                for l in range(-1,1):
                    temp1[i][j] += temp2[i+1+k][j+1+l] * low_pass_filter[k+1][l+1] / low_pass_sum
            temp1[i][j] = temp1[i][j] * low_pass_factor
            temp3[i][j] = img[i][j] * all_pass_factor
            result[i][j] = temp3[i][j] - temp1[i][j]

    return result
```

Images:

Sample2: result5

Problem 2: GEOMETRICAL MODIFICATION

(a)Improve sample3:

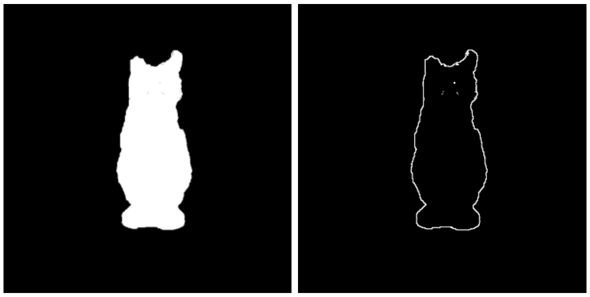
I use 2 method to improve this cute cat.

First, remove the white boundary of this cat.

I make the white part become black and other become white and found out only some small holes exist in this cat's body.

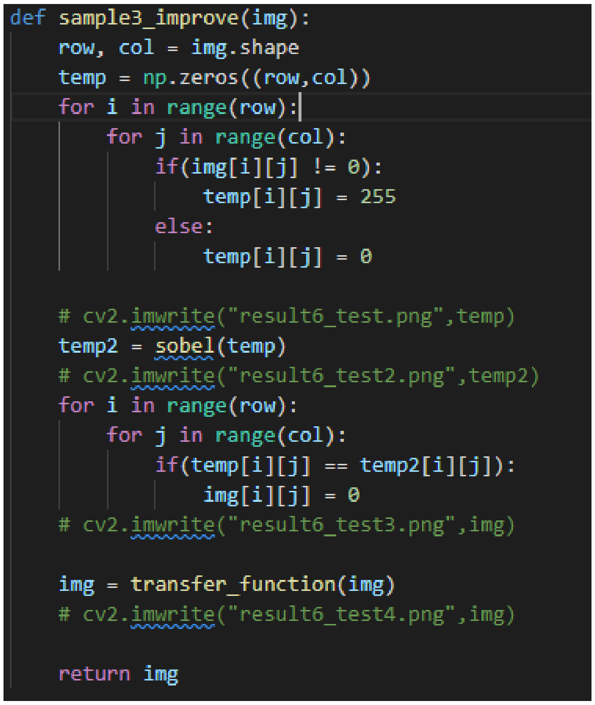
I use sobel to extract the boundary of the image I get from the first step.

And cut it from the original image.

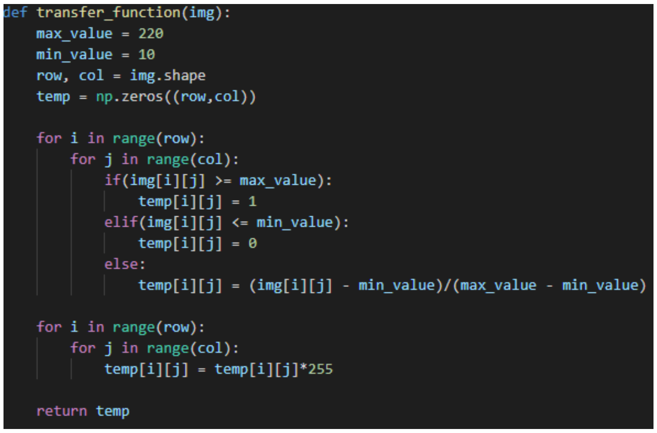


The second method is the transfer function.

I use the HW1 transfer function to improve the contrast.



```
def sample3_improve(img):
    row, col = img.shape
    temp = np.zeros((row,col))
    for i in range(row):
        for j in range(col):
            if(img[i][j] != 0):
                temp[i][j] = 255
            else:
                temp[i][j] = 0

    # cv2.imwrite("result6_test.png",temp)
    temp2 = sobel(temp)
    # cv2.imwrite("result6_test2.png",temp2)
    for i in range(row):
        for j in range(col):
            if(temp[i][j] == temp2[i][j]):
                img[i][j] = 0
    # cv2.imwrite("result6_test3.png",img)

    img = transfer_function(img)
    # cv2.imwrite("result6_test4.png",img)

    return img
```

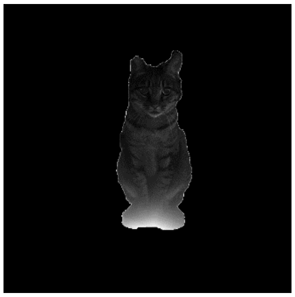


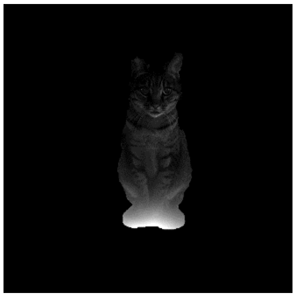
```
def transfer_function(img):
    max_value = 220
    min_value = 10
    row, col = img.shape
    temp = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            if(img[i][j] >= max_value):
                temp[i][j] = 1
            elif(img[i][j] <= min_value):
                temp[i][j] = 0
            else:
                temp[i][j] = (img[i][j] - min_value)/(max_value - min_value)

    for i in range(row):
        for j in range(col):
            temp[i][j] = temp[i][j]*255

    return temp
```

This transfer function is linear, and max value and min value I use are 220, 10

Result:

Sample3 result6





(b)Rotation, Scaling, Translation

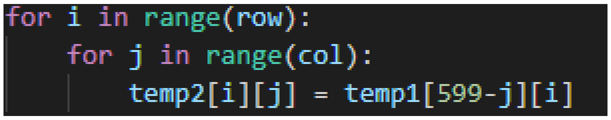
First, I use downsampling as the scaling.

I pick the top-leftmost element in a 2*2 matrix

And shift this cat to right 145 pixels and down 20 pixels

Rotate 90 degree

I only use a simple function to do this part.

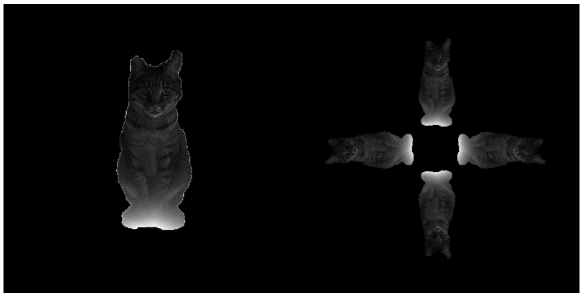


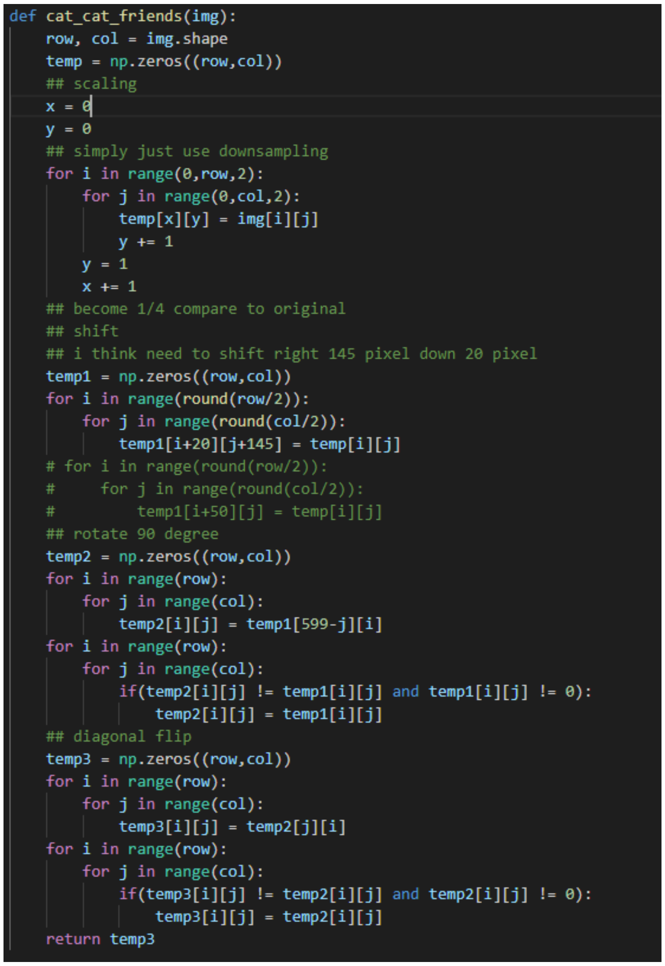
```python
for i in range(row):
    for j in range(col):
        temp2[i][j] = temp1[599-j][i]
```

Then I diagonally flip this picture to get the result.

Result:

Sample3: result7





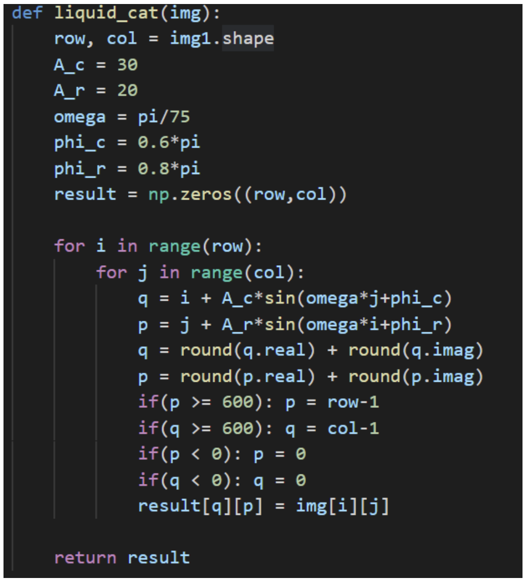
```python
def cat_cat_friends(img):
    row, col = img.shape
    temp = np.zeros((row,col))
    ## scaling
    x = 0
    y = 0
    ## simply just use downsampling
    for i in range(0,row,2):
        for j in range(0,col,2):
            temp[x][y] = img[i][j]
            y += 1
        y = 1
        x += 1
    ## become 1/4 compare to original
    ## shift
    ## i think need to shift right 145 pixel down 20 pixel
    temp1 = np.zeros((row,col))
    for i in range(round(row/2)):
        for j in range(round(col/2)):
            temp1[i+20][j+145] = temp[i][j]
    # for i in range(round(row/2)):
    #     for j in range(round(col/2)):
    #         temp1[i+50][j] = temp[i][j]
    ## rotate 90 degree
    temp2 = np.zeros((row,col))
    for i in range(row):
        for j in range(col):
            temp2[i][j] = temp1[599-j][i]
    for i in range(row):
        for j in range(col):
            if(temp2[i][j] != temp1[i][j] and temp1[i][j] != 0):
                temp2[i][j] = temp1[i][j]
    ## diagonal flip
    temp3 = np.zeros((row,col))
    for i in range(row):
        for j in range(col):
            temp3[i][j] = temp2[j][i]
    for i in range(row):
        for j in range(col):
            if(temp3[i][j] != temp2[i][j] and temp2[i][j] != 0):
                temp3[i][j] = temp2[i][j]
    return temp3
```

(c)Liquid cat

After observing the image, I found out that the line on the original image is actually sin function like.

If I can find the new position for every element, I can make new image close to the sample image.

My parameter and function as below:



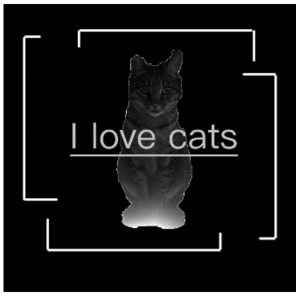
```
def liquid_cat(img):
    row, col = img1.shape
    A_c = 30
    A_r = 20
    omega = pi/75
    phi_c = 0.6*pi
    phi_r = 0.8*pi
    result = np.zeros((row,col))

    for i in range(row):
        for j in range(col):
            q = i + A_c*sin(omega*j+phi_c)
            p = j + A_r*sin(omega*i+phi_r)
            q = round(q.real) + round(q.imag)
            p = round(p.real) + round(p.imag)
            if(p >= 600): p = row-1
            if(q >= 600): q = col-1
            if(p < 0): p = 0
            if(q < 0): q = 0
            result[q][p] = img[i][j]

    return result
```

This method is actually easy to implement, but there are some shorts exist.

1. Some points will not be mapped by this transfer function and use round will lose some information, so the new image will have lots of holes.
2. The shape of new image is close to original, but it's still obviously different from the original one.

Result:

Sample5



result8