SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

IMAGE AND VIDEO PROCESSING

CLASS PROJECT 8

Morphological Operations

Author:
Jie YANG

Supervisor:
JIANHONG SHI

November 27, 2018

CONTENTS

Morphological Operations

Introduction

Morphological operations are important operations in image processing. It used to operate some interested component in the image. It can increase of decrease the interested component area by erosion and dilation, respectively. All the operations need a structural element, which is a small component compare to whole image. Like the window, it can select neighbor pixels of interested pixel as ROI, then compare to original image to determine the values of this ROI.

$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\} \tag{1}$$

The erosion of a set A by a structural element B is defined as the set of all points z such that B, translated by z is still contained in A **Eq.1**.

$$A \oplus B = \left\{ z \middle| (\hat{B})_z \bigcap A \neq \emptyset \right\} \Leftrightarrow \left\{ z \middle| \left[(\hat{B})_z \bigcap A \right] \subseteq A \right\} \tag{2}$$

The dilation of a set A by a structuring element B is defined as **Eq.2**. Of course $A \subseteq A \oplus B$, the dilation expands/dilutes the image.

$$A \circ B = (A \ominus B) \oplus B \Leftrightarrow \bigcup \left\{ (B)_z \middle| (B)_z \subseteq A \right\} \tag{3}$$

Opening is a process of erosion followed by dilation. It has the effect to eliminate small and thin object, breaking the object at thin points and smoothing the boundaries of the objects. The operation is defined as **Eq.3**.

$$A \bullet B = (A \oplus B) \ominus B \Leftrightarrow \bigcup \left\{ (B)_z \middle| (B)_z \bigcap A \neq \emptyset \right\} \tag{4}$$

Closing is a process of dilation followed by erosion. It has the effect of filling small and thin holes, connecting nearby objects and smoothing the boundaries of the objects **Eq.4**.

In this experiment, we will practice four problems. Problem 1 requires to define two structural elements, perform binary dilation, erosion, opening, and closing operations on two noise image. Problem 2 should extract the boundaries of two images. Problem 3 should extract the connected components from original image. Problem 4 require to separate particles based on merged with boundary, overlapping and nonoverlapping.

From this practice, student should know how to use basic morphological operations to process image and how to apply them.

Method

The structural elements are binary image, 255 for white and 0 for dark pixel. The size of SE must be odd.

Problem 1

First padding original image with radius of SE $r = (l-1)/2$. So that the edge pixels can be processed. Finally remove padded boundary so that the image size does not change.

- Dilation

  Padding with 0, then find all bright pixels as seed. For each bright pixel select the window part as the same size of SE and do **OR** operation bit by bit. Finally use the result as the value of window then delete this bright pixel from list until list be null.

- Erosion

  Padding with 255 and choose dark pixels as seed, then reverse the value of SE. Same to dilation select ROI with same size to SE but do **AND** operation bit by bit and use the result as the value of part in erosion image until list be null.

- Opening

  Opening is done by erosion then dilation.

- Closing

  Closing is done by dilation then erosion.

Problem 2

Use 3×3 full bright square as SE. Then erose image with this SE, finally use original image subtract the erosion image to get the boundary image.

Problem 3

Algorithm 1 Search all connected component from binary image

Input: binary image A, structural element E
Output: connected component Y, area A, x center x, y center y

```
Given binary image A, structural element E
Find out all white pixels as seed
while seed ≠ ∅ do
    Find white pixel coordinates in seed
    X_0 = one white pixel in seed
    while X_k ≠ X_{k-1} do
        X_k = (X_{k-1} ⊕ E) ∩ A
    end while
    Y = X_k
    A = the number of white pixels in Y
    x = mean coordinate of white pixels in Y
    y = mean coordinate of white pixels in Y
    Remove connected component from seed
end while
return Y, A, x, y
```

Use 3×3 full bright square as SE. Find out all bright pixels as seed, then choose one pixels to search all of its connected pixels and get the area of this connected area. The area is the number of white pixels. Then remove this connected component from seed. Repeat above procedures until there is no white pixels in the image. So that all connected component be caught and get its area and center point. Center point is the mean value of horizontal axis and vertical axis the component takes. The whole algorithm is listed in **Alg. 1**

The method of how to find out connected component from one pixel is that first dilate the pixel with SE, then find the intersection with original image. Use this intersection to update source pixels. After that dilate all updated source pixels to repeat above steps until there is no change. The whole procedure is show as **Fig.1**.

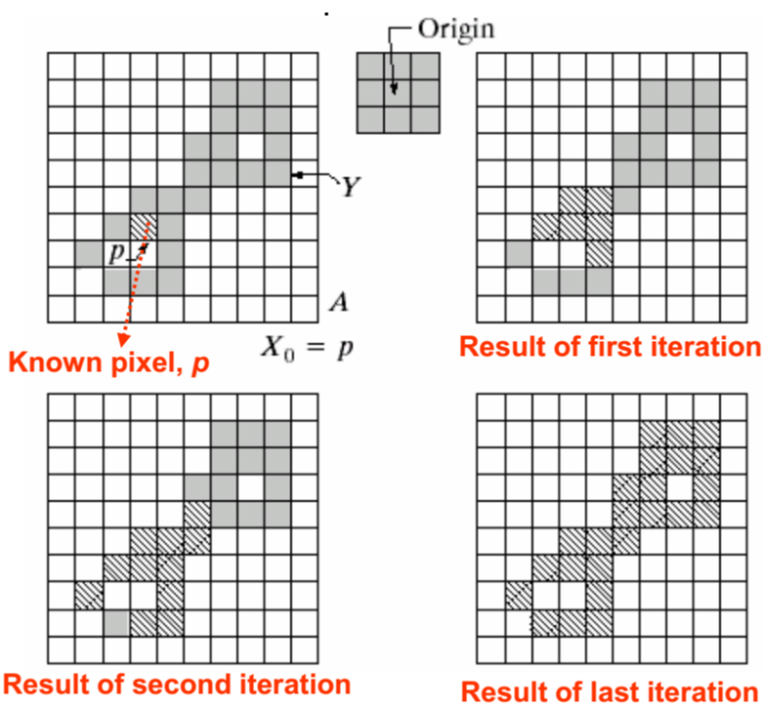


Figure 1: Find connected component

Problem 4

Color the image border pixels the same color as the particles (white). Call the resulting set of border pixels B. Apply the connected component algorithm **Alg. 1**. All connected component that contain elements from B are particles that have merged with the border of the image.

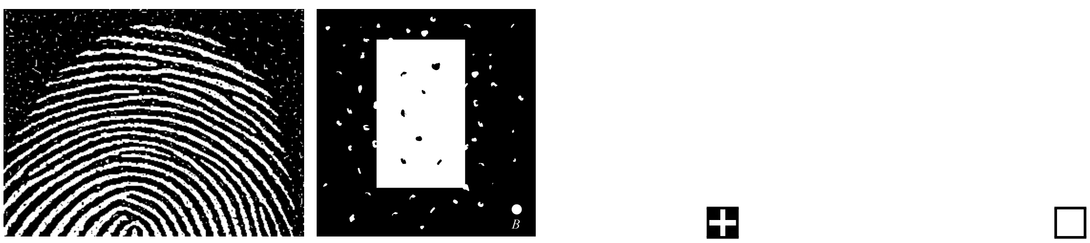
It is given that all particles are of the same size. Determine the area of a single particle; denote the area by A. Eliminate from the image the particles that were merged with the border of the image. Apply the connected component algorithm. Count the number of pixels of each component. A components is then designated as a single particle if the number of pixels is less than or equal to $k * A (k > 1)$, where k is a coefficient to account for variation in size due to noise.

Subtract from the image single particles and the particles that have merged with the border, and the remaining particles are overlapping particles.

Results

Problem 1

The first structural element (SE) is a cross element, whose vertical and vertical that pass center is light while other pixels are are zero. Second SE is full bright SE, all pixels are bright. They are both 5×5 structural element.**Fig.2**

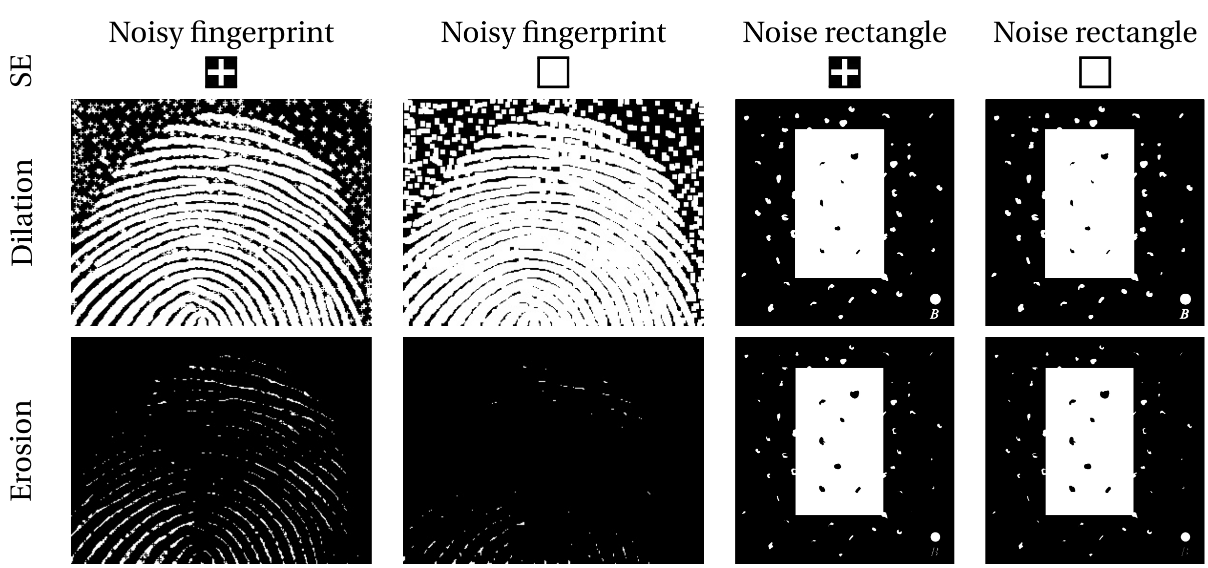


(a) Noisy fingerprint (b) Noise rectangle (c) 5×5 cross SE (d) 5×5 SE

Figure 2: Original images and structural elements

For dilation, both SE increase the size of white component. In noisy fingerprint image. First SE has more clear grooves. And for single point, first SE dilation dilate it to cross shape while second one is square.

For erosion, there is only little bright component in noisy fingerprint with second SE while first SE remain more bright component. It is similar to noise rectangle.**Fig.3**



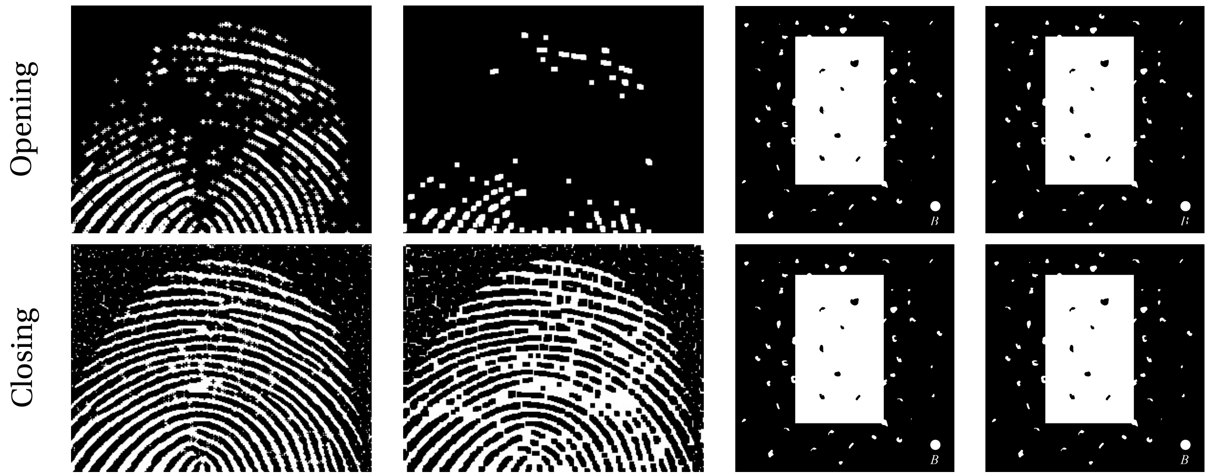


Figure 3: Basic morphological Operations

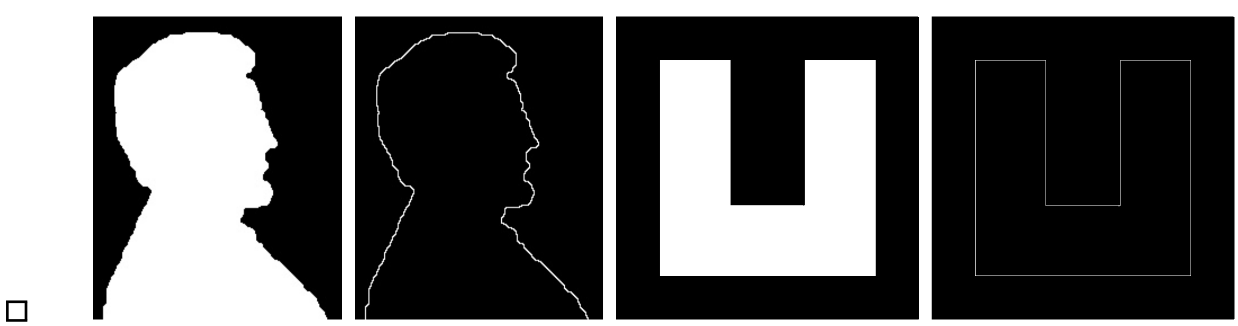
For opening operation, it remain more bright component that erosion operation the line width also thicker than erosion. Compare to first SE, second SE still remains less.

For closing operation, there are more bright component. The small noisy point around fingerprint also appeared. Compare to second SE, first SE has better performance. It separate the ridges in finger and attenuate the background noise. The ridges od second fingerprint is thicker and has several holes.

Compare to fingerprint image. the difference between two Se in noise rectangle image is not clear enough.

Problem 2

The structural element applied there is a square SE whose size is 3×3. **Fig.4-a**I t is clear to see that after boundary extraction, the boundary of man appears. The width is is 1 pixel. **Fig.4-c** For U image, there is a sunk in the inner right corner, it is still be catched. **Fig.4-e**



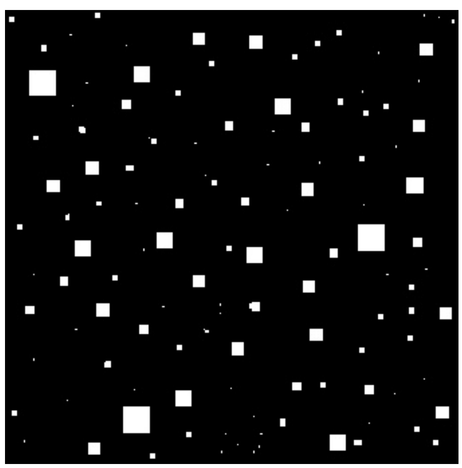
(a) SE (b) Original licoln (c) Licoln boundary (d) Original U (e) U boundary

Figure 4: Boundary extraction

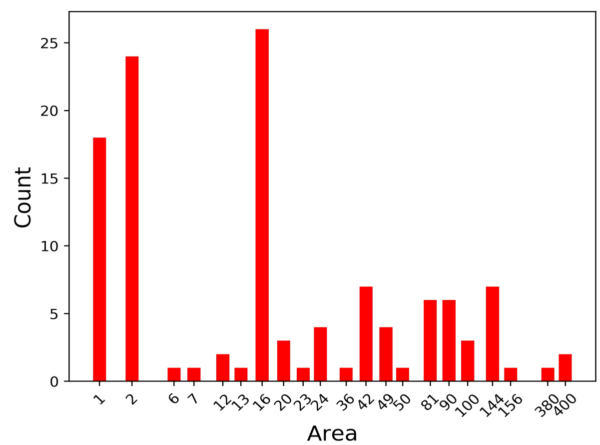
Problem 3

The structural element applied there is a square SE whose size is 3×3. **Fig.5-a** From the original image, there are several squares in the image. **Fig.5-b** The area histogram of original image indicated that 4×4 square takes most amount. Second and third is 2 pixels and 1 pixel point. The biggest square 20×20. The counting number also shows that all connect component be extracted. **Fig.5-c** There are totally 120 connect component in the image. According to the distribution of center point of each connected components, their mean values close to half size of image, which indicated that the connect component distribute nearly to uniform in image. **Fig.5-c,d** The separated images of each size connect component is in **Fig.7** and the center point of each of them are in **Tab.1**.

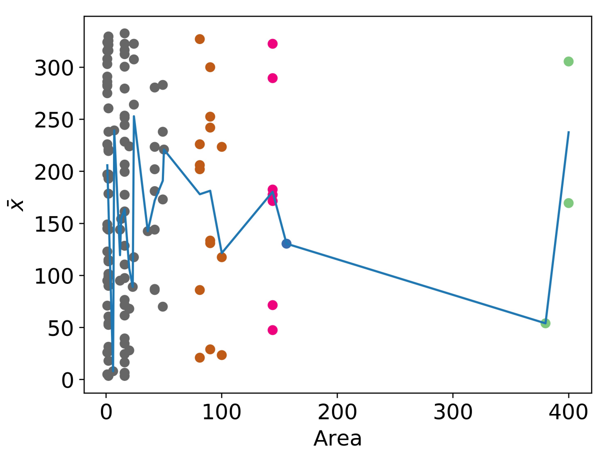
(a) SE



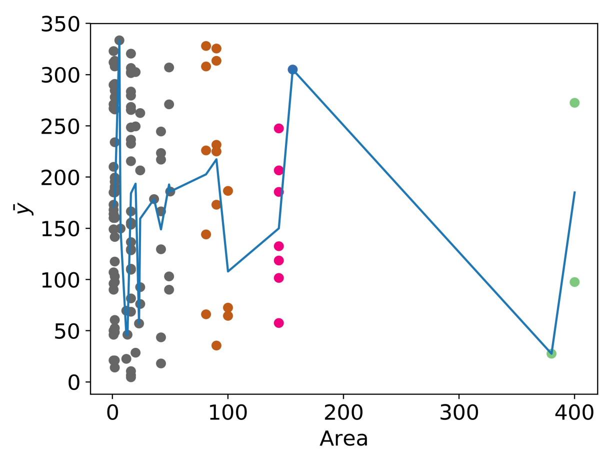
(b) Original connected image



(c) Connected area histogram



(d) Connected component center at x



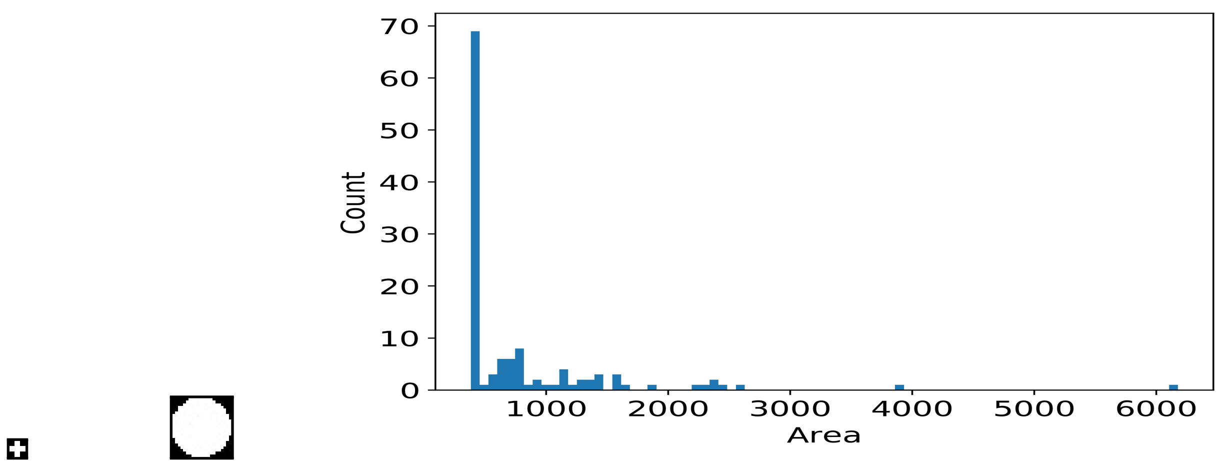
(e) Connected component center at y

Figure 5: Boundary extraction

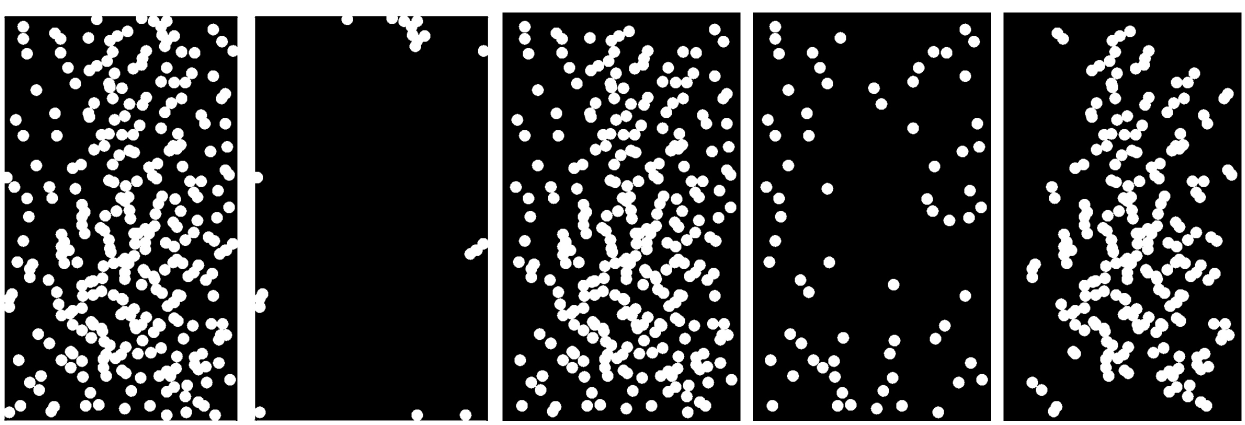
Problem 4

The structural element applied there is a cross SE whose size is 3×3. ffreffig:p4a The radius of sigle particle is 11 pixels.**Fig.6-b** From the connect area histogram, there are close to 70 single particles in the image. The biggest connected component takes larger than 6000 pixels.**Fig.6-c**

Compare to original image **Fig.6-d**, all merged with boundary particles be extracted.**Fig.6-e** There is no connect particles in nonoverlapping image**Fig.6-g** and all overlapped particles are in last image.**Fig.6-h** All the images indicate that the separation result is very well. All connect components are 4 pixels connected.



(a) SE (b) Single particle (c) Connect area histogram



(d) Original image (e) Boundary merged (f) Without boundary (g) Nonoverlapping (h) Overlapping

Figure 6: Bubbles on black background particle

Discussion

When you perform erosion, one needs to pad the object and the structuring element to rectangular arrays. During erosion process, the structuring element erodes the input image A at its boundaries. So, erosion shrinks the input image.

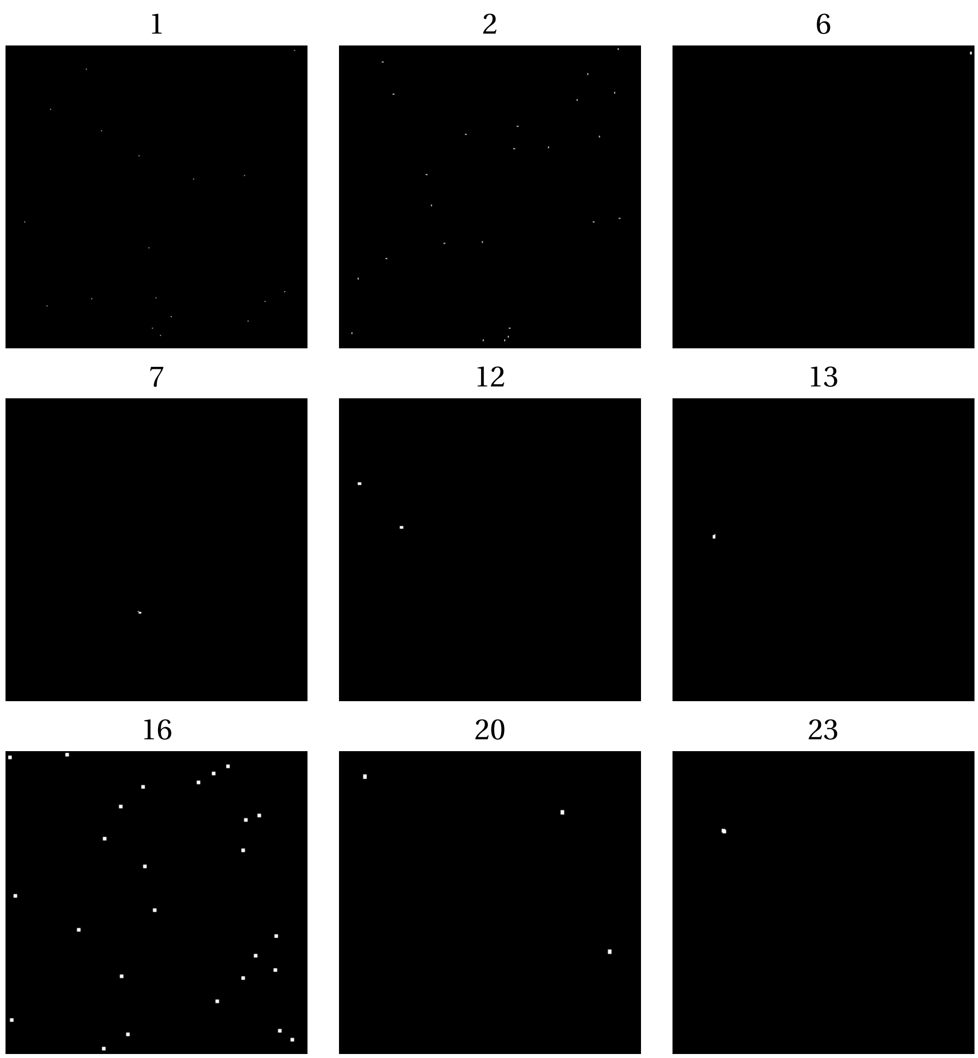
Dilation operation grows or thickens objects in a binary image, that is, increasing the boundary thickness by half side of the structuring element. So, dilation and erosion are opposite operation, dilation is the thickening process and erosion is the thinning process.

The SE should be select based to specific problem. Morphological operations match the SE in original image. For example, if you only want to select N_4 neighbor pixels, SE should be N_4 rather full.

In connected component search procedure, if just use the definition to select ROI and compare, it takes too much time. Because for each pixels, it require to search image. Use modified algorithm, it speed much.

Supplementary

Supplementary images and tables:



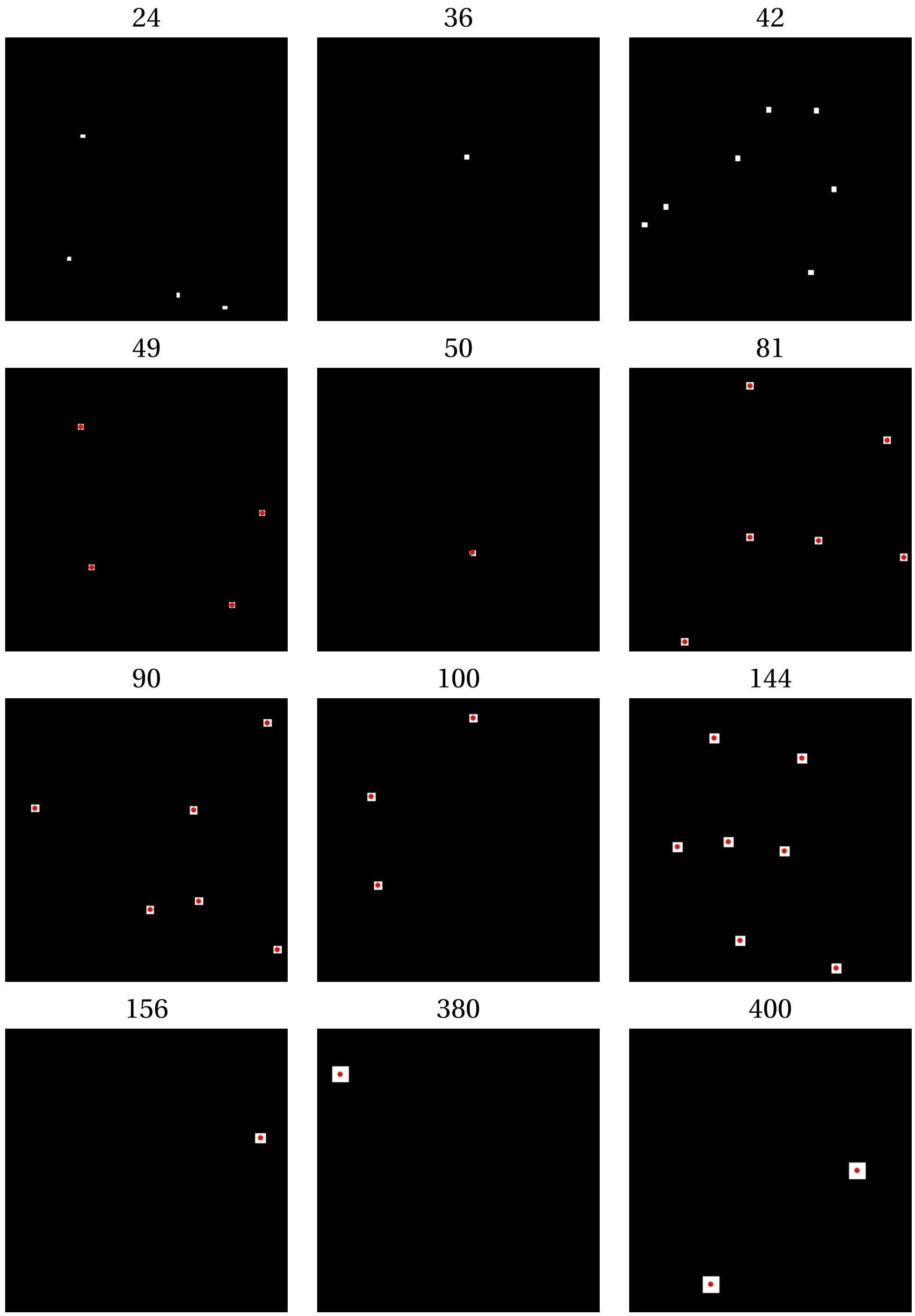


Figure 7: Connect components in connected image

Index	Area	$\bar{x}$	$\bar{y}$	Index	Area	$\bar{x}$	$\bar{y}$
0	1	123.0	149.0	60	16	332.5	109.5
1	1	303.0	185.0	61	16	97.5	110.5
2	1	308.0	271.0	62	16	6.5	4.5
3	1	95.0	107.0	63	16	34.5	215.5
4	1	226.0	160.0	64	16	16.5	248.5
5	1	286.0	290.0	65	16	161.5	10.5
6	1	26.0	90.0	66	16	39.5	153.5
7	1	316.0	164.0	67	16	24.5	232.5
8	1	283.0	96.0	68	16	61.5	128.5
9	1	282.0	168.0	69	16	110.5	265.5
10	1	71.0	50.0	70	16	71.5	283.5
11	1	145.0	267.0	71	16	76.5	268.5
12	1	324.0	173.0	72	16	128.5	155.5
13	1	149.0	210.0	73	20	224.0	302.5
14	1	197.0	21.0	74	20	68.0	249.5
15	1	5.0	323.0	75	20	28.0	28.5
16	1	275.0	312.0	76	23	89.0	57.0
17	1	291.0	46.0	77	24	322.5	262.5
18	2	99.0	141.5	78	24	307.5	206.5
19	2	101.5	291.0	79	24	117.5	92.5
20	2	260.5	21.0	80	24	264.083333	76.083333
21	2	113.5	234.0	81	36	142.5	178.5
22	2	115.0	195.5	82	42	181.0	244.5
23	2	221.0	117.5	83	42	144.0	129.5
24	2	219.5	160.0	84	42	202.0	43.5
25	2	197.0	284.5	85	42	223.5	18.0
26	2	144.0	97.5	86	42	87.0	223.5
27	2	193.0	313.5	87	42	86.0	166.5
28	2	178.5	103.0	88	42	280.5	217.0
29	2	238.0	52.5	89	49	238.0	103.0
30	2	90.0	199.5	90	49	70.0	90.0
31	2	329.5	185.0	91	49	173.0	307.0
32	2	31.5	278.0	92	49	283.0	271.0
33	2	3.5	312.0	93	50	220.920000	185.86
34	2	329.5	161.0	94	81	86.0	308.0
35	2	60.5	266.0	95	81	21.0	144.0
36	2	54.0	60.5	96	81	327.0	66.0
37	2	52.5	308.0	97	81	206.0	226.0
38	2	325.5	189.0	98	81	202.0	144.0

Continued on next page

Index	Area	$\bar{x}$	$\bar{y}$	Index	Area	$\bar{x}$	$\bar{y}$
39	2	18.0	48.5	99	81	226.0	328.0
40	2	316.0	190.5	100	90	252.5	173.0
41	2	321.5	14.0	101	90	133.5	225.0
42	6	8.0	333.5	102	90	300.0	325.5
43	7	239.285714	149.714286	103	90	131.0	35.5
44	12	95.0	22.5	104	90	242.0	231.5
45	12	144.0	69.5	105	90	29.0	313.5
46	13	154.307692	46.076923	106	100	23.5	186.5
47	16	322.5	320.5	107	100	117.5	64.5
48	16	253.5	265.5	108	100	223.5	72.5
49	16	206.5	302.5	109	144	322.5	247.5
50	16	251.5	129.5	110	144	182.5	185.5
51	16	316.5	136.5	111	144	71.5	206.5
52	16	312.5	306.5	112	144	289.5	132.5
53	16	228.5	279.5	113	144	171.5	118.5
54	16	300.5	6.5	114	144	47.5	101.5
55	16	279.5	236.5	115	144	177.5	57.5
56	16	244.5	301.5	116	156	130.5	305.0
57	16	177.5	166.5	117	380	54.0	27.5
58	16	199.5	81.5	118	400	305.5	97.5
59	16	3.5	68.5	119	400	169.5	272.5

Table 1: All connected components in connected image

This is the code used in this project.

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
'''
 @File   : lab8.py
 @Author: Yangjie
 @license : Copyright(C), SUSTech,Shenzhen,China
 @Contact : yangj3@mail.sustc.edu.cn
 @Date   : 2018/11/10
 @IDE : PyCharm
 @Desc   :
 '''
import cv2
import numpy as np
from matplotlib import pyplot as plt
from copy import deepcopy as dcp
import pandas as pd


```

```
class IMG:
    def __init__(self, name, mark=None):
        self.path = 'D:\graduated\Image_process\lab\PGM_images\\'
        self.savepath = 'D:\graduated\Image_process\lab\lab_report\...
                                    lab8\imagesave\\'
        self.name = name
        self.prop = '.pgm'
        self.mark = mark
        # self.img=None

    def load(self):
        self.imapath = self.path + self.name + self.prop
        self.img = np.float64(cv2.imread(self.imapath, 0))
        self.save(self.img, 'original')
        return self.img

    def save(self, img, mark=None, flag=0):
        if flag:
            img = cv2.equalizeHist(np.uint8(img))
        self.mark = mark
        savepath = self.savepath + self.name + '_' + self.mark + '....
                                    jpg'
        cv2.imwrite(savepath, img)
        return img

    def disp(self, winName, img, sizeflag=cv2.WINDOW_NORMAL):

        img = cv2.equalizeHist(np.uint8(img))
        if sizeflag == 1:
            sizeflag = cv2.WINDOW_AUTOSIZE
        cv2.namedWindow(winName, sizeflag)
        cv2.imshow(winName, img)
        cv2.waitKey(0)
        cv2.destroyWindow(winName)
        return img

    def psave(self, img, mark=None, cb=0):  # shown image in windows ...
                                    and save
        fig = plt.gcf()
        plt.imshow(img, cmap='gray')
        if cb:
            plt.colorbar()
        plt.xticks([]), plt.yticks([])
        savepath = self.savepath + self.name + '_' + mark + '.jpg'
        fig.savefig(savepath, dpi=500, bbox_inches='tight')
        plt.close()

    def fsave(self, fig, mark=None):  # save plot fihiure
        # plt.tick_params(labelsize=20)
        # plt.xticks([]), plt.yticks([])
        savepath = self.savepath + self.name + '_' + mark + '.jpg'
```

```
        fig.savefig(savepath, dpi=500, bbox_inches='tight')
        plt.close()

    def plthist(self, img, mark):
        font2 = {'family': 'Times New Roman', 'weight': 'normal', '...
                                    size': 25}
        img = np.uint8(img)
        fig = plt.gcf()
        plt.hist(img.ravel(), 256);
        plt.xlabel('Intensity ', font2)
        plt.ylabel('Count ', font2)
        self.fsave(fig, mark)
        plt.close()


def sfft(img):
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    return fshift


def isfft(fshift):
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    return img_back


def cal_R(x, y, img):
    N = img.shape[0]
    M = img.shape[1]
    u = x - M / 2
    v = N / 2 - y
    R = np.sqrt(u ** 2 + v ** 2)
    return R


class Morphology:
    def __init__(self):
        self.name = 'Morphology'

    def erosion(self, img, EM):  # EM =1
        EM_inv = cv2.bitwise_not(EM)
        up_len = (EM.shape[0] - 1) // 2
        right_len = (EM.shape[1] - 1) // 2
        cmb_img = cv2.copyMakeBorder(img, up_len, up_len, right_len, ...
                                    right_len, cv2.BORDER_CONSTANT,...
                                     value=1)
        dst = np.uint8(np.ones(cmb_img.shape) * 255)
        X, Y = np.where(img == 0)
        X = X.tolist()
        Y = Y.tolist()
```

```
        while X:
            x = X.pop()
            y = Y.pop()
            # dst[x:x + EM.shape[0], y:y + EM.shape[1] ]=cv2....
                                        bitwise_and(EM_inv,img[x:x + EM...
                                        .shape[0], y:y + EM.shape[1] ])
            temp = cv2.bitwise_and(EM_inv, cmb_img[x:x + EM.shape[0],...
                                         y:y + EM.shape[1]])

            old = dst[x:x + EM.shape[0], y:y + EM.shape[1]]
            dst[x:x + EM.shape[0], y:y + EM.shape[1]] = cv2....
                                        bitwise_and(temp, old)
        dst = dst[up_len:-1 * up_len, right_len:-1 * right_len]
        return np.uint8(dst)

    def dilation(self, img, EM):  # EM is 1, foreground is 1
        up_len = (EM.shape[0] - 1) // 2
        right_len = (EM.shape[1] - 1) // 2
        cmb_img = cv2.copyMakeBorder(img, up_len, up_len, right_len, ...
                                        right_len, cv2.BORDER_CONSTANT,...
                                         value=0)
        # print(cmb_img.shape)
        dst = np.uint8(np.zeros(cmb_img.shape))
        X, Y = np.where(img == 255)
        X = X.tolist()
        Y = Y.tolist()
        while X:
            x = X.pop()
            y = Y.pop()
            # dst[x:x + EM.shape[0], y:y + EM.shape[1] ]=cv2....
                                        bitwise_or(EM,cmb_img[x:x + EM....
                                        shape[0], y:y + EM.shape[1] ])
            temp = cv2.bitwise_or(EM, cmb_img[x:x + EM.shape[0], y:y ...
                                        + EM.shape[1]])
            old = dst[x:x + EM.shape[0], y:y + EM.shape[1]]
            dst[x:x + EM.shape[0], y:y + EM.shape[1]] = cv2....
                                        bitwise_or(temp, old)
        dst = dst[up_len:-1 * up_len, right_len:-1 * right_len]
        return np.uint8(dst)

    def opening(self, img, EM):
        e = self.erosion(img, EM)
        dst = self.dilation(e, EM)
        return dst

    def closing(self, img, EM):
        e = self.dilation(img, EM)
        dst = self.erosion(e, EM)
        return dst

    def boundary(self, img, EM):
```

```
        dst = np.uint8(img - self.erosion(img, EM))
        return dst

    def connect_detec_point(self, A, EM, x, y,visual=0):
        Xk_1 = A * 0
        Xk = dcp(Xk_1)
        Xk[x, y] = 255;
        if visual:
            winName = 'find connect'
            cv2.namedWindow(winName, cv2.WINDOW_NORMAL)
        while (Xk != Xk_1).any():
            if visual:
                cv2.imshow(winName, Xk)
                cv2.waitKey(1)
            # |cv2.destroyWindow(winName)
            Xk_1 = Xk
            temp = self.dilation(Xk_1, EM)
            Xk = cv2.bitwise_and(dcp(temp), dcp(A))
        X, Y = np.where(Xk == 255)
        mx = np.mean(X)
        my = np.mean(Y)
        area = len(X)
        data = [X, Y, area, mx, my]
        statics = pd.DataFrame([data], columns=['X', 'Y', 'Area', 'mx...
                                               ', 'my'])
        if visual:
            cv2.waitKey(1)
            cv2.destroyWindow(winName)

        return (statics, Xk)

    def connect_detec(self, A, EM, Seed,visual=0):
        Connect_set = A * 0
        result = pd.DataFrame()
        if visual:
            winName1 = 'Seed'
            cv2.namedWindow(winName1, cv2.WINDOW_NORMAL)
        while Seed.any():
            if visual:
                cv2.imshow(winName1, Seed)
            X, Y = np.where(Seed == 255)
            statics, Conect = self.connect_detec_point(A, EM, X[0], Y...
                                               [0])
            result = result.append(statics, ignore_index=True)
            Conect_inv = cv2.bitwise_not(Conect)
            Seed = cv2.bitwise_and(Seed, Conect_inv)
        print('Catch %d connected component' % len(result))
        if visual:
            cv2.waitKey(1)
            cv2.destroyWindow(winName1)
        return result
```

```python


## -------------- P1----------------
print('\n---Problem 1----\n')
imnameset = ['noisy_fingerprint', 'noise_rectangle']
#EM1 = np.uint8(np.ones((3, 3)))*255
EM2 = np.uint8(np.ones((5, 5)))*255
EM1 = np.uint8(np.zeros((5, 5)))*255

EM1[2,:] = 255
EM1[:,2] = 255


EMset =[EM1,EM2]
for k,EM in enumerate(EMset):
    for imname in imnameset:
        I = IMG(imname)
        img = np.uint8(I.load())


        I.save(EM, 'EM_'+str(k))
        M = Morphology()

        print(imname+'\t dilation...')
        temp=M.dilation(img, EM)
        I.save(temp,'dilation'+'_EM_'+str(k))

        print(imname + '\t erosion...')
        temp = M.erosion(img, EM)
        I.save(temp, 'erosion'+'_EM_'+str(k))

        print(imname + '\t opening...')
        temp = M.opening(img, EM)
        I.save(temp, 'opening'+'_EM_'+str(k))

        print(imname + '\t closing...\n')
        temp = M.closing(img, EM)
        I.save(temp, 'closing'+'_EM_'+str(k))
print('\n==== Problem 1 done ====\n')

## -------------- P2----------------
print('\n---Problem 2----\n')
imnameset=['licoln','U']
for imname in imnameset:
        I = IMG(imname)
        img = np.uint8(I.load())

        EM = np.uint8(np.ones((3, 3)))*255
        I.save(EM, 'EM')
        M = Morphology()

```

```
        print(imname + '\t boundary...')
        temp = M.boundary(img, EM)
        I.save(temp, 'boundary')
print('\n==== Problem 2 done ====\n')

## -------------- P3----------------
print('\n---Problem 3----\n')
imnameset = ['connected']
for imname in imnameset:
    I = IMG(imname)
    img = np.uint8(I.load())
    EM = np.uint8(np.ones((3, 3)))*255
    I.save(EM, 'EM')
    M = Morphology()

    result = M.connect_detec(img, EM, img)
    result = result.sort_values(by='Area')
    #  save data frame
    result.to_csv(I.savepath + 'result.csv')
    result.index = np.arange(len(result))
    # write latex formate
    latex = result.to_latex(longtable=True, escape=False)
    with open(I.savepath + 'table.txt', 'w') as f:
        f.write(latex)

    data = result[['Area', 'mx', 'my']]
    latex = data.to_latex(longtable=True, escape=False)
    with open(I.savepath + 'data.txt', 'w') as f:
        f.write(latex)

    # get information of each area
    print('\t Generate images   ...  \n ')
    sx = list(set(data['Area'].values))
    sx.sort()
    y = []
    smx = []
    smy = []
    for s in sx:
        select = data[data['Area'] == s]
        mdata = np.mean(select)
        amount = len(select)
        y.append(amount)
        smx.append(mdata[1])
        smy.append(mdata[2])
        #  plot each area set images
        select = result[result['Area'] == s]
        cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR) * 0
        for index, row in select.iterrows():
            xset = row['X']
            yset = row['Y']
            for i, j in zip(list(xset), list(yset)):
```

```
                cimg[i, j] = [255, 255, 255]
            if s > 45:
                cv2.circle(cimg, (int(row['my']), int(row['mx'])), 3,...
                                              (0, 0, 255), -1)
        I.save(cimg, str(s))

    # plot images
    print('\t Plot images ... \n')
    px = list(map(lambda x: np.log(x) / np.log(2), sx))  # range(len(...
                                              x))
    px = np.array(px) + range(len(sx))
    fig1 = plt.gcf()
    plt.bar(px, y, fc='r', tick_label=list(map(lambda x: str(x), sx))...
                                              )
    plt.xlabel('Area', fontsize=15)
    plt.ylabel('Count', fontsize=15)
    plt.tick_params(labelsize=10)
    plt.xticks(rotation=45)
    I.fsave(fig1, mark='count')

    fig = plt.gcf()
    x = list(data['Area'].values)
    y = list(data['mx'].values)
    plt.scatter(x, y, c=x, cmap=plt.cm.Accent_r)
    plt.xlabel('Area', {'size': 15})
    plt.ylabel(r'$\bar{x}$', {'size': 15})
    plt.plot(sx, smx)
    plt.tick_params(labelsize=15)
    I.fsave(fig, mark='x')

    fig = plt.gcf()
    x = list(data['Area'].values)
    y = list(data['my'].values)
    plt.scatter(x, y, c=x, cmap=plt.cm.Accent_r)
    plt.plot(sx, smy)
    plt.xlabel('Area', {'size': 15})
    plt.ylabel(r'$\bar{y}$', {'size': 15})
    plt.tick_params(labelsize=15)
    I.fsave(fig, mark='y')

print('\n==== Problem 3 done ====\n')

## -------------- P4----------------
print('\n==== Problem 4  ====\n')
imnameset = ['bubbles_on_black_background']
for imname in imnameset:
    I = IMG(imname)
    img = np.uint8(I.load())
    plt.imshow(img, 'gray')
    particle = img[225:247, 96:118]  # clsoed bouble
    I.save(particle, 'particle')
```

```
    p_area = np.sum(particle) / 255
    EM = np.uint8(np.ones((3, 3))) * 255
    I.save(EM, 'full_EM')
    EM[0, 0] = 0
    EM[0, -1] = 0
    EM[-1, 0] = 0
    EM[-1, -1] = 0
    I.save(EM, 'EM')

    M = Morphology()

    # (a)
    print('\n particle  merged with the boundary ... \n')
    add_boundary_img = dcp(img)
    add_boundary_img[-1, :] = 255
    add_boundary_img[0, :] = 255
    add_boundary_img[:, 0] = 255
    add_boundary_img[:, -1] = 255

    statics, add_boundary = M.connect_detec_point(add_boundary_img, ...
                                                  EM, 0, 0)
    boundary = cv2.bitwise_and(add_boundary, img)
    I.save(boundary, 'boundary')
    img2 = img - boundary
    I.save(img2, 'seperate+connect')

    print('\n seperate  particles  ... \n')
    result = M.connect_detec(img2, EM, img2)
    select = result[result['Area'] < 1.1 * p_area]
    separate = img * 0
    for index, row in select.iterrows():
        xset = row['X']
        yset = row['Y']
        for i, j in zip(list(xset), list(yset)):
            separate[i, j] = 255
    I.save(separate, 'seperate')

    print('\n connected particles  ... \n')
    connect = img2 - separate
    I.save(connect, 'connect')

    area = result['Area'].values
    plt.close()
    fig = plt.gcf()
    plt.hist(area, 80)
    plt.xlabel('Area', {'size': 15})
    plt.ylabel('Count', {'size': 15})
    plt.tick_params(labelsize=15)
    I.fsave(fig, mark='area_hist')

print('\n==== Problem 4 done ====\n')
```

Code Listing 1: Python code for image processing