Practical Assignment № 1

basic methods for images segmentation into semantic areas.

Instructing Professor :Sergei Shavetov & Andrei Zhdanov
Author :Xu Miao

April 26, 2022

Contents

1 Introduction

§1.1 Student information

- ⋆ Name: Xu Miao
- ⋆ ITMO Number: 293687
- ⋆ HDU Number: 19322103

§1.2 Purpose.

Study of the basic methods for images segmentation into semantic areas.

§1.3 Introduction

In this report, I will complete the content of the following sections:

1) **Binarization.**
   a) Choose **upper and lower binarization thresholds** based on the image
   b) Binarize the image
2) **Segmentation 1.**
   a) Perform the image containing the face(s) segmentation according to the **Weber principle.**
   b) Perform the image segmentation based on the skin color and try **different formula**s with different photo illumination conditions.
3) **Segmentation 2.**
   a) Perform image segmentation in the **CIE Lab** color space by the nearest neighbors method
   b) Perform image segmentation in the **CIE Lab** color space by the *k*-means method
4) **Segmentation 3.**
   a) Perform texture segmentation of the image
   b) Evaluate at least three parameters of the selected textures, determine which class the textures belong to

2 Notations

Table 2.1: Notation used in this report

Symbol	Definition
I	image brightness
R	The luminance component of the R channel of the image
G	The luminance component of the G channel of the image
B	The luminance component of the B channel of the image
m	mean value
s	Standard deviation
R	Relative smoothness
E	entropy

⋆ Other notations instructions will be given in the text.

3 Solution of problem

§3.1 Binarization.

§3.1.i Theorem.

he simplest way to segment an image into two classes (background and object pixels) is **binarization**. **Binarization** can be performed by **thresholding** or by **double thresholding**.

thresholding
Thresholding is divided into upper thresholding and lower thresholding:
upper thresholding:

$$I_{new}(x,y) = \begin{cases} 0, I(x,y) \leqslant t \\ 1, I(x,y) > t \end{cases} \tag{3.1}$$

lower thresholding:

$$I_{new}(x,y) = \begin{cases} 1, I(x,y) \leqslant t \\ 0, I(x,y) > t \end{cases} \tag{3.2}$$

where I - source image, I_{new} – binarized image, t– binarization threshold.

double thresholding
Double thresholding uses the following formula:

$$I_{new}(x,y) = \begin{cases} 0, I(x,y) \leqslant t_1 \\ 1, t_1 < I(x,y) \leqslant t_2, \\ 0, I(x,y) > t_2, \end{cases} \tag{3.3}$$

where I - source image, I_{new} - binarized image, t_1 and t_2– upper and lower binarization thresholds.

Commonly used methods for finding thresholds
In practical applications, the most commonly used methods for automatic threshold calculation are **Otsu method** and **adaptive threshold method**(This lab report focuses on these two methods):

1) Otsu method

   Calculation of the optimal threshold t by statistical method Otsu which divides all pixels into two classes 1 and 2 . This method minimizing the variance within each class $\sigma_1^2(t)$ and $\sigma_2^2(t)$, and maximizing the variance between classes.

2) Adaptive threshold method

   Adaptive methods that do not work with the entire image, but only with its frag-

ments. Such approaches are often used when working with images that represent non-uniformly illuminated objects.

§3.1.ii Solution

1. upper & lower thresholding:

1.a Selection of the original image

Select the original image as shown in Fig.3.1:

Fig. 3.1. Vincent van Gogh: ”Sunflowers”

1.b Image processing result

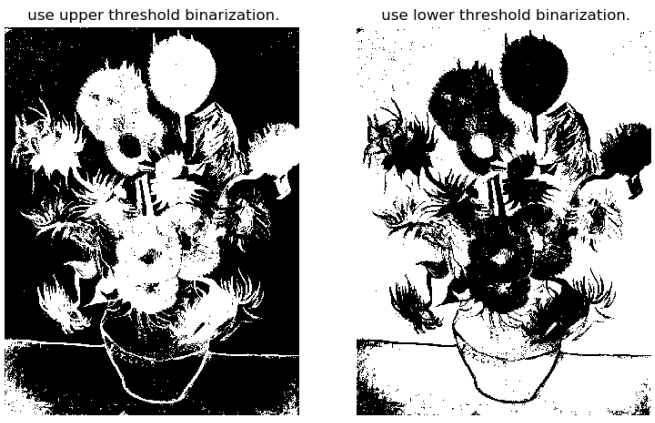

Fig. 3.2. Upper & lower threshold binarization image processing results(Use OTSU to select the threshold)

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ Upper Thresholding and Lower Thresholding If **the same thresholding method** is selected, the **segmentation area is the same**, **but reversed** (i.e. the black and white parts of the two images are reversed)
- ⋆ The upper threshold binarization or the lower threshold binarization can be selected according to **the target image style**

2. Double threshold binarization:

1.a Selection of the original image

In order to compare the characteristics of different selection threshold methods (this experiment report chooses to explore the most frequently used **OTSU** and **adaptive methods**), I selected two original images with obvious characteristics as shown in Fig.3.4 and Fig.3.5 below:

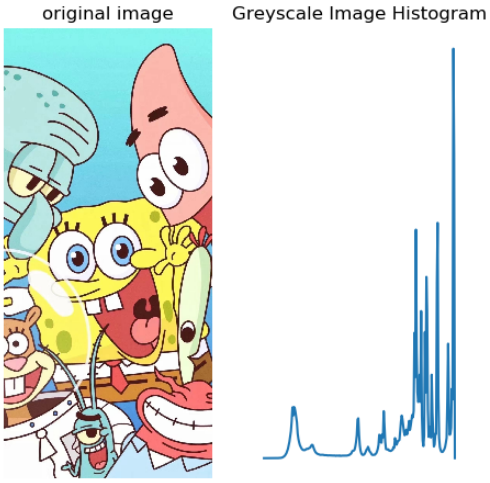

Fig. 3.3. The original image with a relatively uniform distribution of brightness histograms

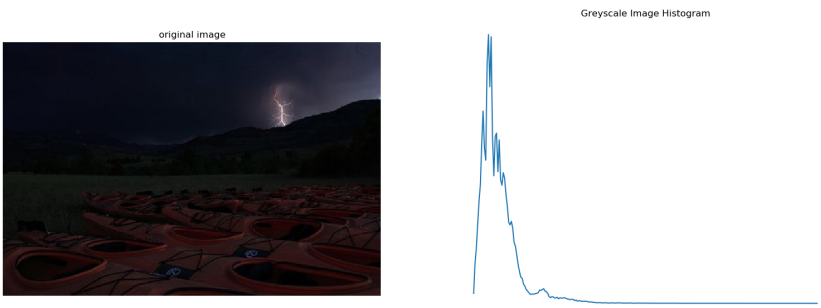

Fig. 3.4. Original image with uneven distribution of brightness histogram

1.b Image processing result

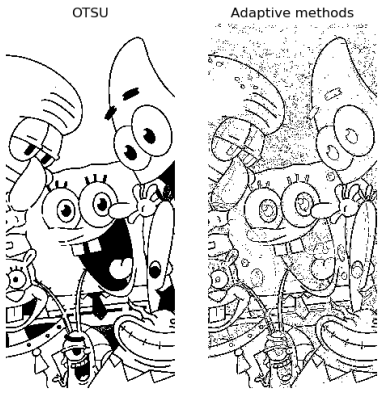

Fig. 3.5. Double threshold binarization process image result1

Fig. 3.6. Double threshold binarization process image result2

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ For the brightness distribution is relatively uniform, the brightness histogram shows that the image similar to ”double peaks” OTSU method performs better, while the adaptive method appears noise effect.
- ⋆ For images with severely uneven brightness distribution, the OTSU method performs poorly and basically cannot segment correctly, while the adaptive method can handle such images well.
- ⋆ In the actual binarization application, the threshold should be selected according to the actual scene and the characteristics of the target. If the illumination is uneven, adaptive threshold segmentation should be selected. If there is no obvious uneven illumination and there are obvious double peaks in the histogram, OTSU should be selected.

§3.2 Segmentation 1.

§3.2.i Theorem.

Image segmentation based on Weber principle

The algorithm is designed for segmentation of grayscale images. Weber principle assumes that **the human eye does not perceive well the difference in gray levels between** $I(n)$ **and** $I(n) + W(I(n))$, where $W(I(n))$– Weber function, n– class number, I– piecewise nonlinear grayscale function. The Weber function can be calculated using the formula:

$$W(I) = \begin{cases} 20 - \frac{12I}{88}, 0 \leqslant I \leqslant 88 \\ 0,002(I - 88)^2, 88 < I \leqslant 138 \\ \frac{7(I-138)}{117} + 13, 138 < I \leqslant 255. \end{cases} \tag{3.4}$$

The segmentation algorithm consists of the following steps:

1. Initialization initial conditions: first class number $n = 1$, grayscale level $I(n) = 0$.
2. Calculation value $W(I(n))$ according to the Weber formula and value assign $I(n)$ to all pixels whose intensity is in the range $[I(n), I(n) + W(I(n))]$.
3. Search for pixels with intensity higher $G = I(n) + W(I(n)) + 1$. If found, then increase the class number $n = n + 1$, assign $I(n) = G$ and go to the second step. Otherwise, finish the algorithm. The image will be segmented into n classes with the intensity $W(I(n))$.

Segmentation of RGB images by skin color

For the segmentation of skin in images, there are the following analytical descriptions for different situations:

1. **at uniform day light illumination:**

$$\begin{cases} R > 95 \\ G > 40 \\ B > 20 \\ \max R, G, B - \min R, G, B > 15 \\ |R - G| > 15 \\ R > G \\ R > B \end{cases} \tag{3.5}$$

2. **under flash light or daylight lateral illumination:**

$$\begin{cases} R > 220 \\ G > 210 \\ B > 170 \\ |R - G| \leqslant 15 \\ G > B \\ R > B \end{cases} \tag{3.6}$$

3. **using normalized RGB values:**

$$\begin{cases} r = \frac{R}{R+G+B}, \\ g = \frac{G}{R+G+B}, \\ b = \frac{B}{R+G+B}, \\ \frac{r}{g} > 1,185, \\ \frac{rb}{(r+g+b)^2} > 0,107, \\ \frac{rg}{(r+g+b)^2} > 0,112. \end{cases} \tag{3.7}$$

§3.2.ii Solution

1.Image segmentation based on Weber principle

1.a Selection of the original image

Select a photo with a human face as shown below:

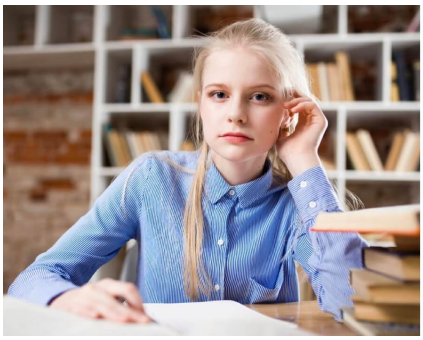

Fig. 3.7. The original image

1.b Image processing result

Because Weber deals with grayscale images, the original image is grayscaled and then segmented using Weber's principle. The result is shown in Fig.3.8 below:

Fig. 3.8. Weber's principle segmentation result graph

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ From Fig.3.8, it can be seen that the images segmented by Weber's principle are basically indistinguishable to the human eye, and there is **less information lost in the image**, so it can be used in image compression algorithms
- ⋆ The disadvantage of Weber's principle is that it can **only** be applied to **grayscale images**

2.Segmentation of RGB images by skin color

1.a Selection of the original image

In order to explore the application of different formulas in different occasions, I chose the following two pictures under two lighting conditions as the original pictures:

Fig. 3.9. Face photo under uniform sunlight

Fig. 3.10. face photo in flash

1.b Image processing result

Using the formulas (3.5) to (3.7) for different scenarios to segment the skin of the two photos, respectively, the results are shown in Fig.3.11 and Fig.3.12:

Fig. 3.11. Skin segmentation results of photos under uniform sunlight

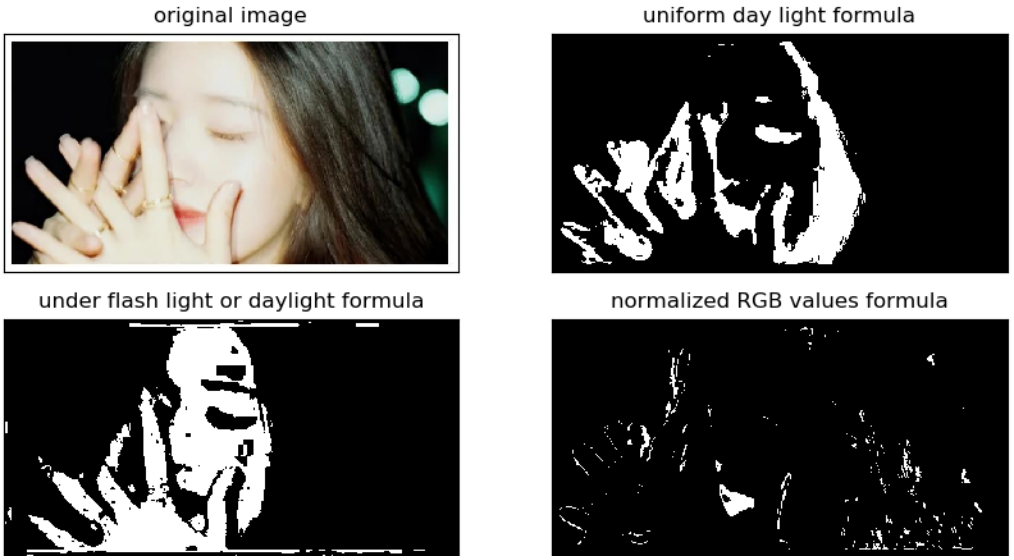

Fig. 3.12. Skin segmentation results of photos under flash

1.c full source code

You can click ⋆here⋆ to see the full code for this part.

1.d Comments

- ⋆ The uniform sunlight formula and the formula under the flash perform well in their corresponding situations and can segment the skin part more accurately, but in the opposite case only a small part of the skin can be identified.
- ⋆ From the two images of the experiment, the normalized RGB formula can only identify part of the skin. It does not perform as well as the first two formulas under the corresponding conditions

§3.3 Segmentation 2.

§3.3.i Theorem.

CIE Lab color space

As shown in Fig.3.7 below, in the Lab color space, the value of lightness is separated from the value of the chromatic components of the color (hue, saturation). The lightness is given by the L coordinate, which can range from 0 (dark) to 100 (light). The chromatic component of a color is given by two Cartesian coordinates a (means the color position in the range from green (−128) to red (127)) and b (means the color position in the range blue (−128) to yellow (127)). A binary image is obtained with zero coordinates a and b.

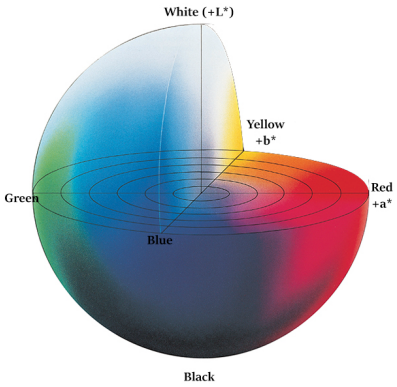

Fig. 3.13. CIE Lab Color Space

CIE Lab color space : Nearest neighbors method

The implementation idea of the nearest neighbor algorithm is as follows:

1. Interactively select to split into N colors and $C_i(i = 1, \cdots, N)$ to represent each color
2. Calculate the center point of the color $C_i(i = 1, \cdots, N)$ in the lab space a_i, b_i The average value of the coordinates in the small circle is used as the reference color $color_i = (a_i, b_i)(i = 1, \cdots, N)$
3. For each pixel in the image, calculate the Euclidean distance between it and the reference color $color_i = (a_i, b_i)(i = 1, \cdots, N)$, Compare and split it into the nearest color $color_i$ class

CIE Lab color space : k-means method

The algorithm idea of using the k-means algorithm to segment the image color is as follows:

1. Consider all points on the image as sample points on the (a,b) plane in CIE Lab space, and find N center points according to the distance between the samples $color_i = (a_i, b_i)(i = 1, \cdots, N)$ divides the sample set into N clusters. Let the points

in the cluster be as close as possible, and the distance between the clusters should be as large as possible.

§3.3.ii Solution

0. Selection of the original image

In order to compare the sensitivity of each algorithm for different situations, I selected two representative photos of ocean balls shown below, containing two and three colors respectively:

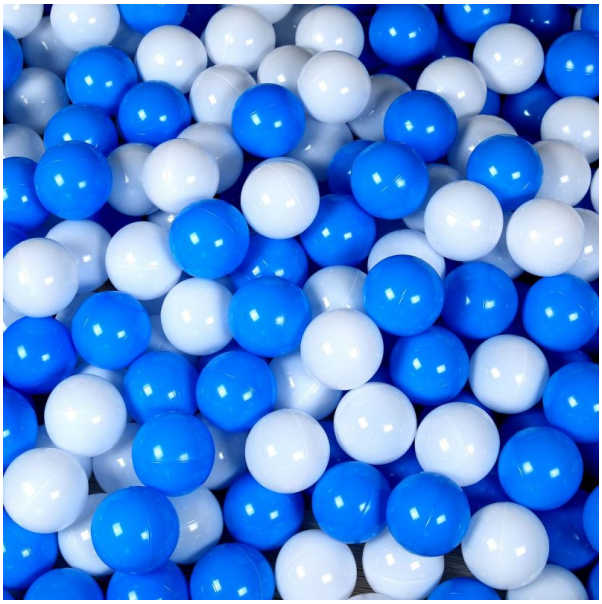

Fig. 3.14. Original image : Two colored ocean balls mixed together

Fig. 3.15. Original image : Three colored ocean balls mixed together

1. CIE Lab color space : Nearest neighbors method

1.a Image processing result

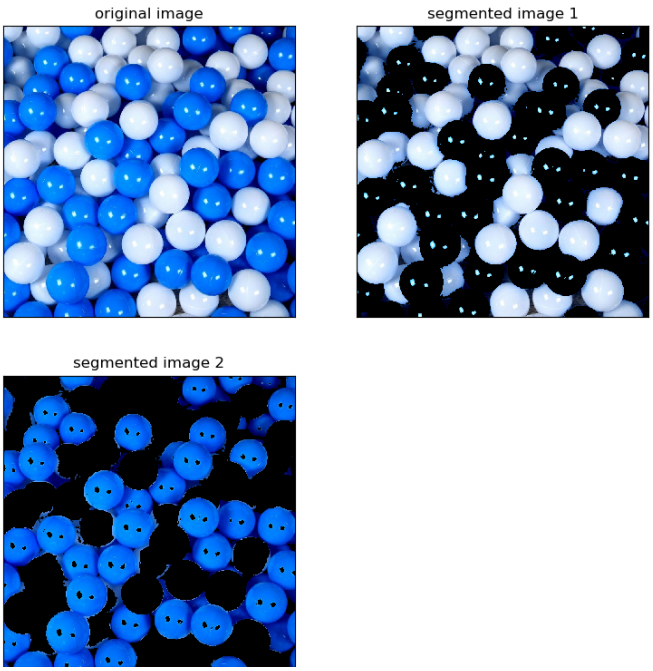

Fig. 3.16. Nearest Neighbor Method: Segmentation Results 1

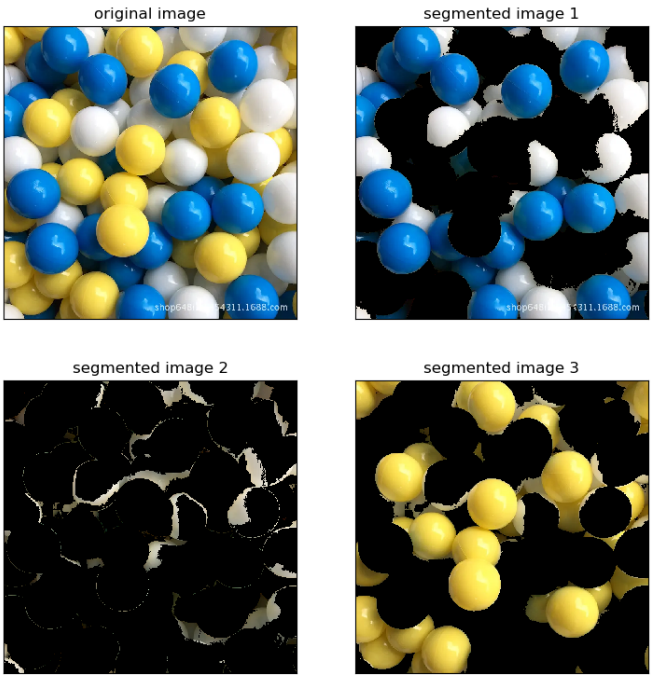

Fig. 3.17. Nearest Neighbor Method: Segmentation Results 2

1.b full source code

You can click ⋆here⋆ to see the full code for this part.

2. CIE Lab color space : k-means method

2.a Image processing result

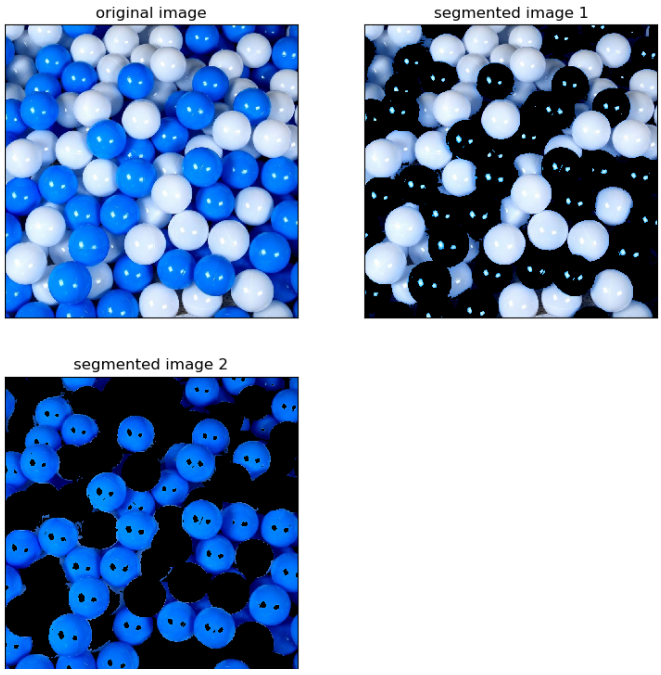

Fig. 3.18. k-means Method: Segmentation Results 1

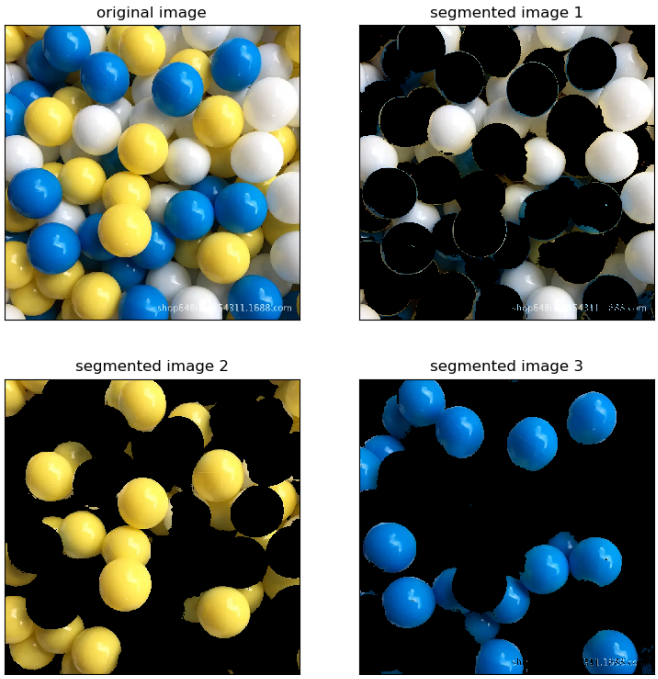

Fig. 3.19. k-means Method: Segmentation Results 2

2.b full source code

You can click ⋆here⋆ to see the full code for this part.

3. Comments

- ⋆ When there are few classification types, both the nearest neighbor and kmeans methods have better performance and can better segment images.
- ⋆ As the number of categories increases, the accuracy of the nearest neighbor method decreases. It can be seen from Fig. 3.17 that when the number of colors in the picture reaches three, the segmentation accuracy decreases seriously.
- ⋆ From the experimental results, it seems that using the k-means method for classification segmentation is better than the nearest neighbor method

§3.4 Segmentation 3.

§3.4.i Theorem.

Texture segmentation
In texture segmentation, three main methods are used to describe texture: statistical, structural, and spectral.I have selected the following statistical parameters to perform texture segmentation on images:
We will consider the image intensity I as a random variable z, which corresponds to the distribution probability $p(z)$ calculated from the image histogram.

1. m — **mean value of a random variable** z

$$m = \sum_{i=0}^{L-1} z_i p(z_i) \tag{3.8}$$

2. s — **Standard deviation of random variable** z

$$s = \sigma(z) \tag{3.9}$$

3. R — **relative smoothness**

$$R = 1 - \frac{1}{1 + \sigma^2(z)} \tag{3.10}$$

4. **histogram symmetry characteristic :** E — **local entropy of the image**

$$E = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i) \tag{3.11}$$

§3.4.ii Solution

0. Selection of the original image

In order to compare the characteristics of the statistical parameters of each comparison segmentation texture, I selected two different types of images with two different textures as shown below:

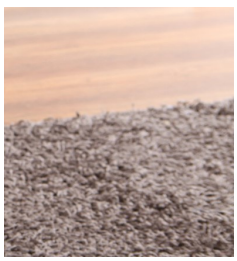

Fig. 3.20. Carpet on wooden floor

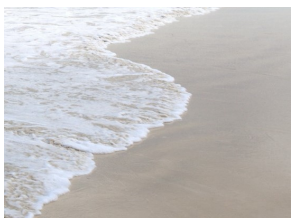

Fig. 3.21. Waves on the beach

1. Image texture segmentation: mean m as texture feature value

1.a Image processing result

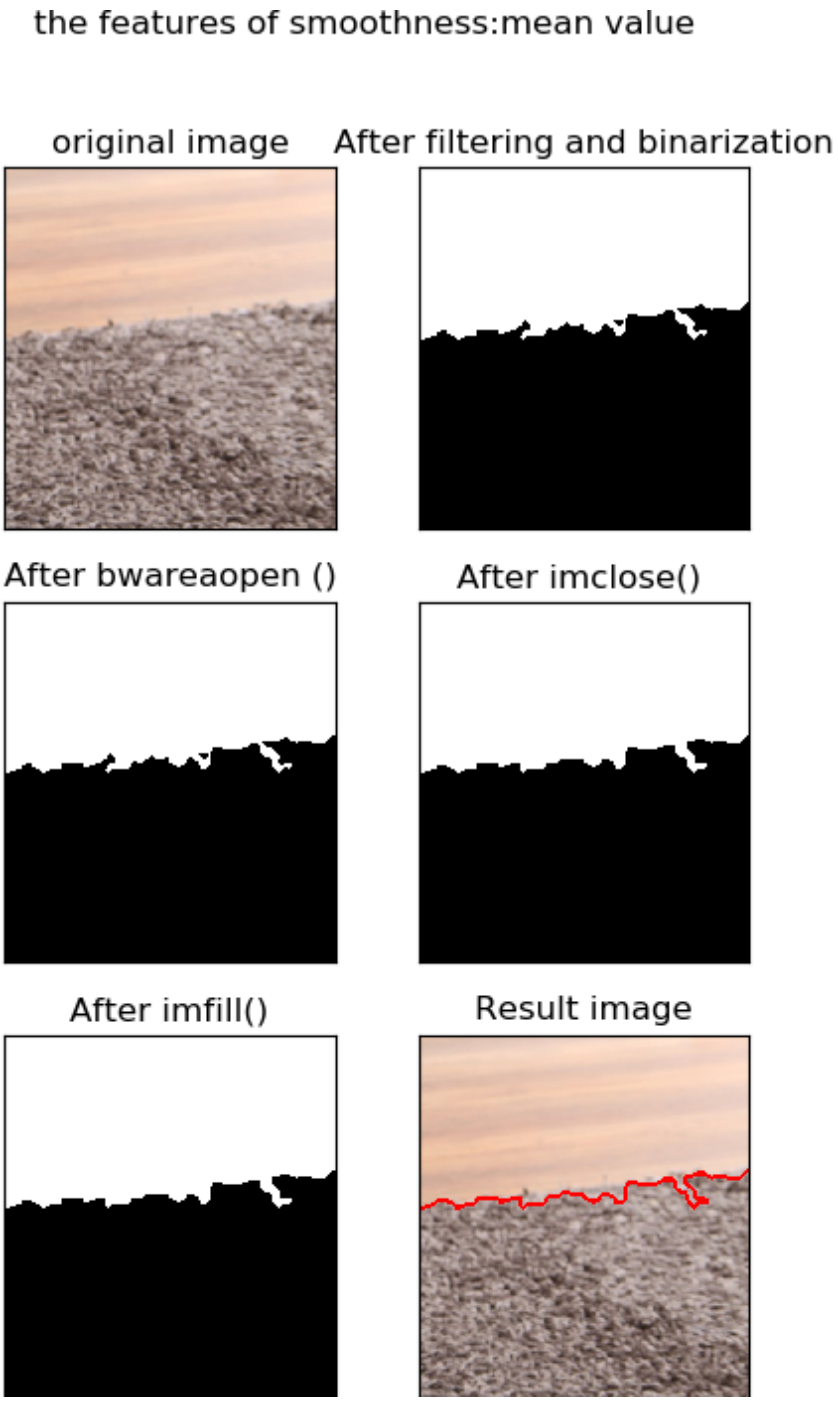

Fig. 3.22. Original image 1 segmentation result

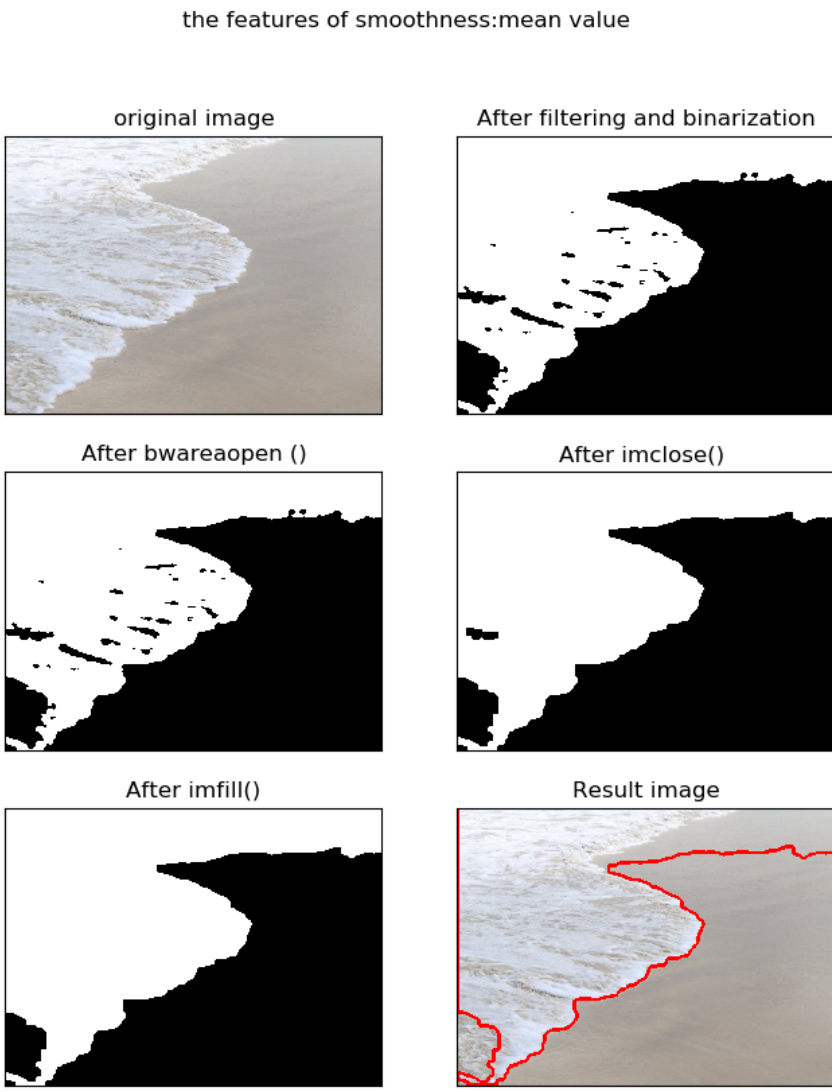

Fig. 3.23. Original image 2 segmentation result

1.b full source code

You can click ⋆here⋆ to see the full code for this part.

2. Image texture segmentation: Standard deviation σ as texture feature

2.a Image processing result

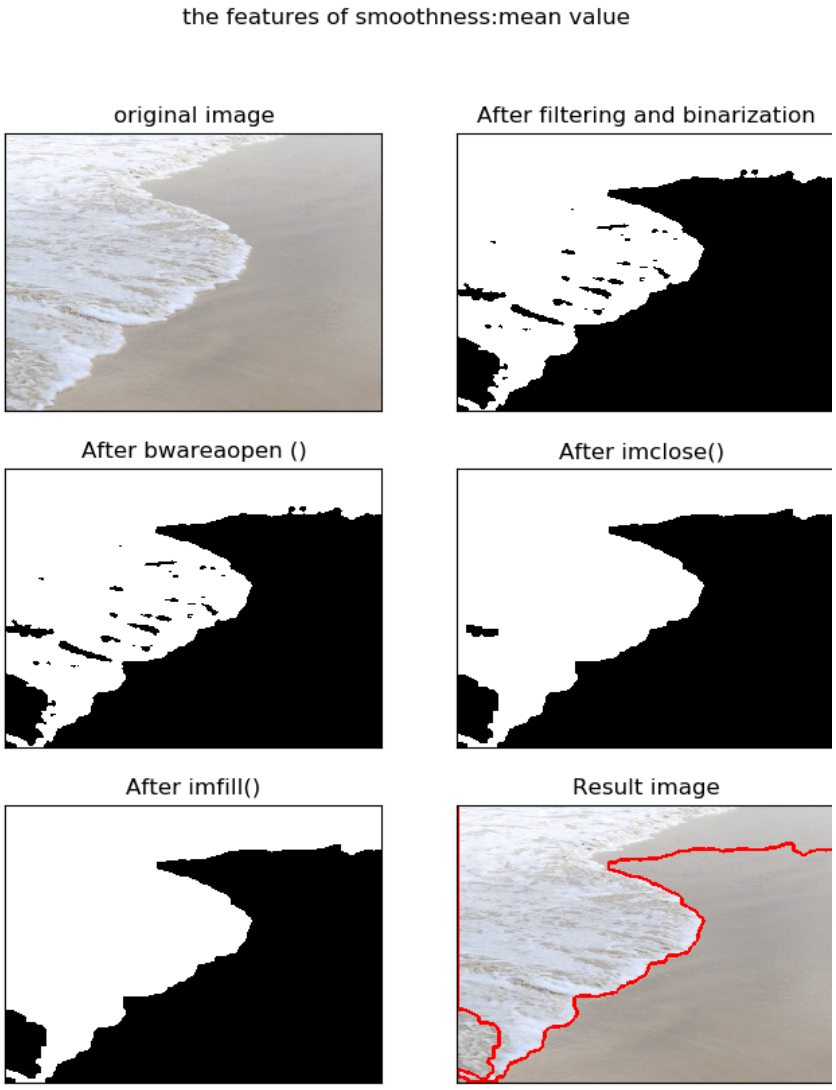

Fig. 3.24. Original image 1 segmentation result

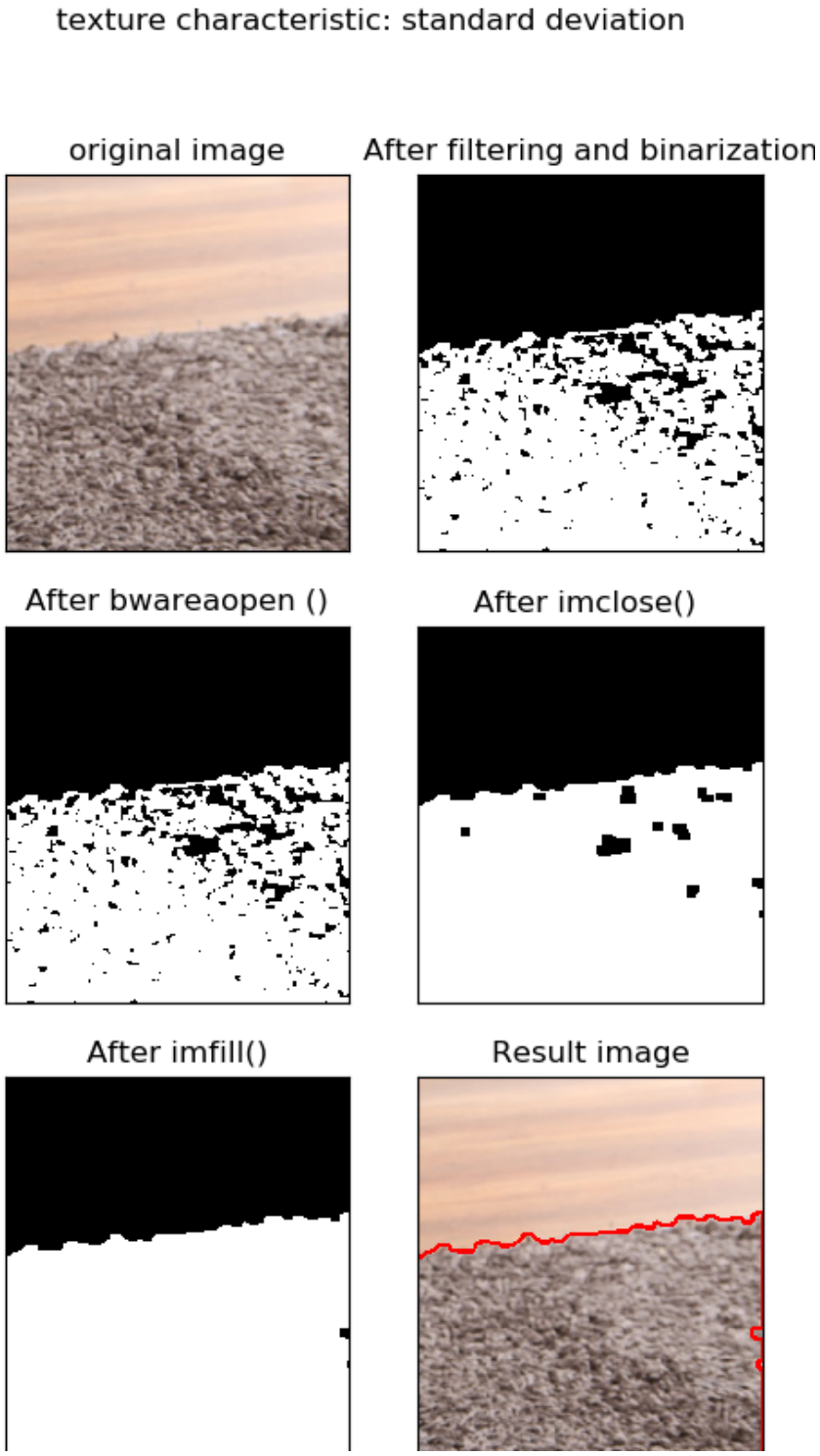

Fig. 3.25. Original image 2 segmentation result

2.b full source code

You can click ⋆here⋆ to see the full code for this part.

3. Image texture segmentation: Relative Smoothness R as texture feature

3.a Image processing result

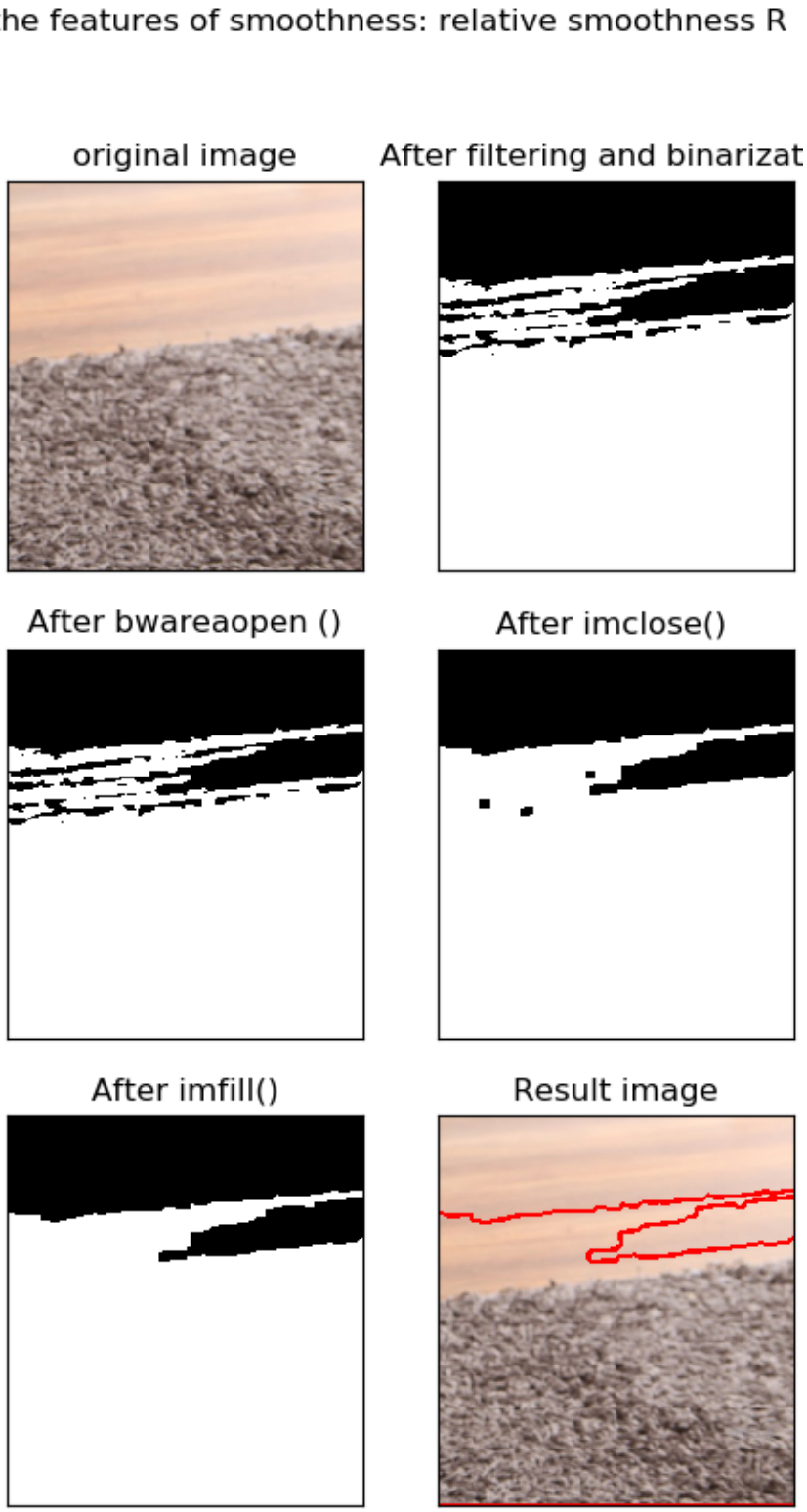

Fig. 3.26. Original image 1 segmentation result

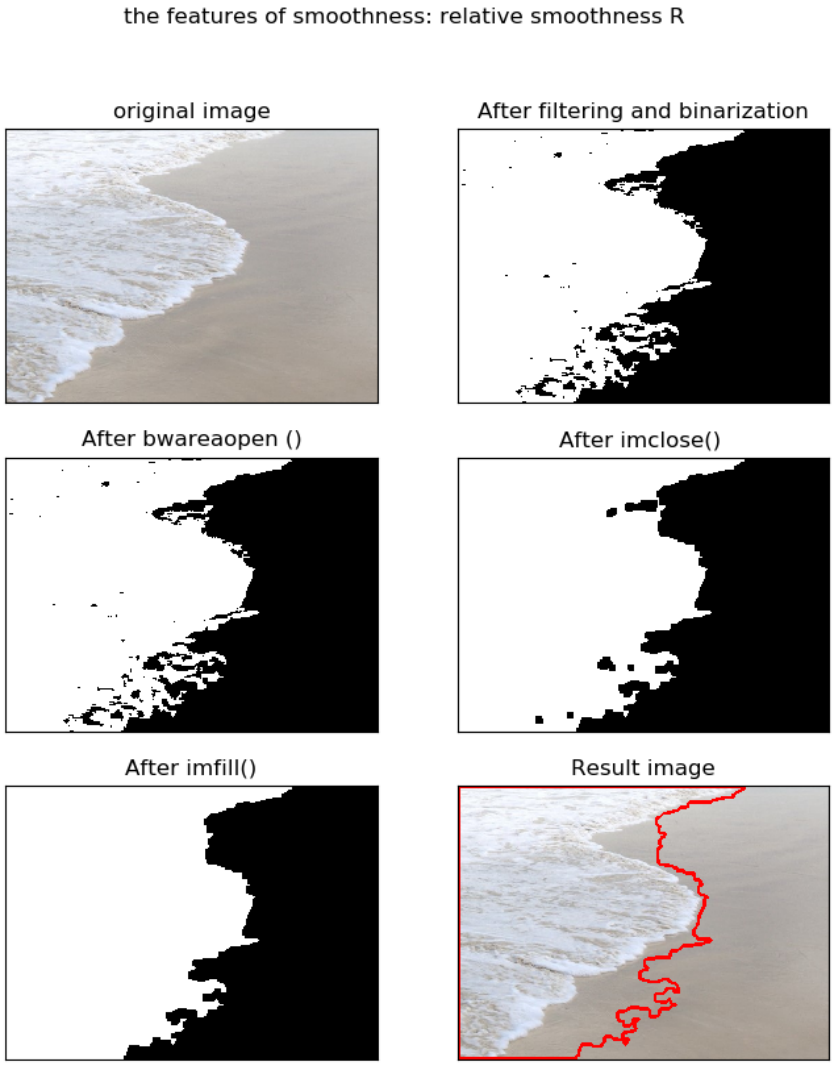

Fig. 3.27. Original image 2 segmentation result

3.b full source code

You can click ⋆here⋆ to see the full code for this part.

4. Image texture segmentation: Local Entropy E as texture feature

4.a Image processing result

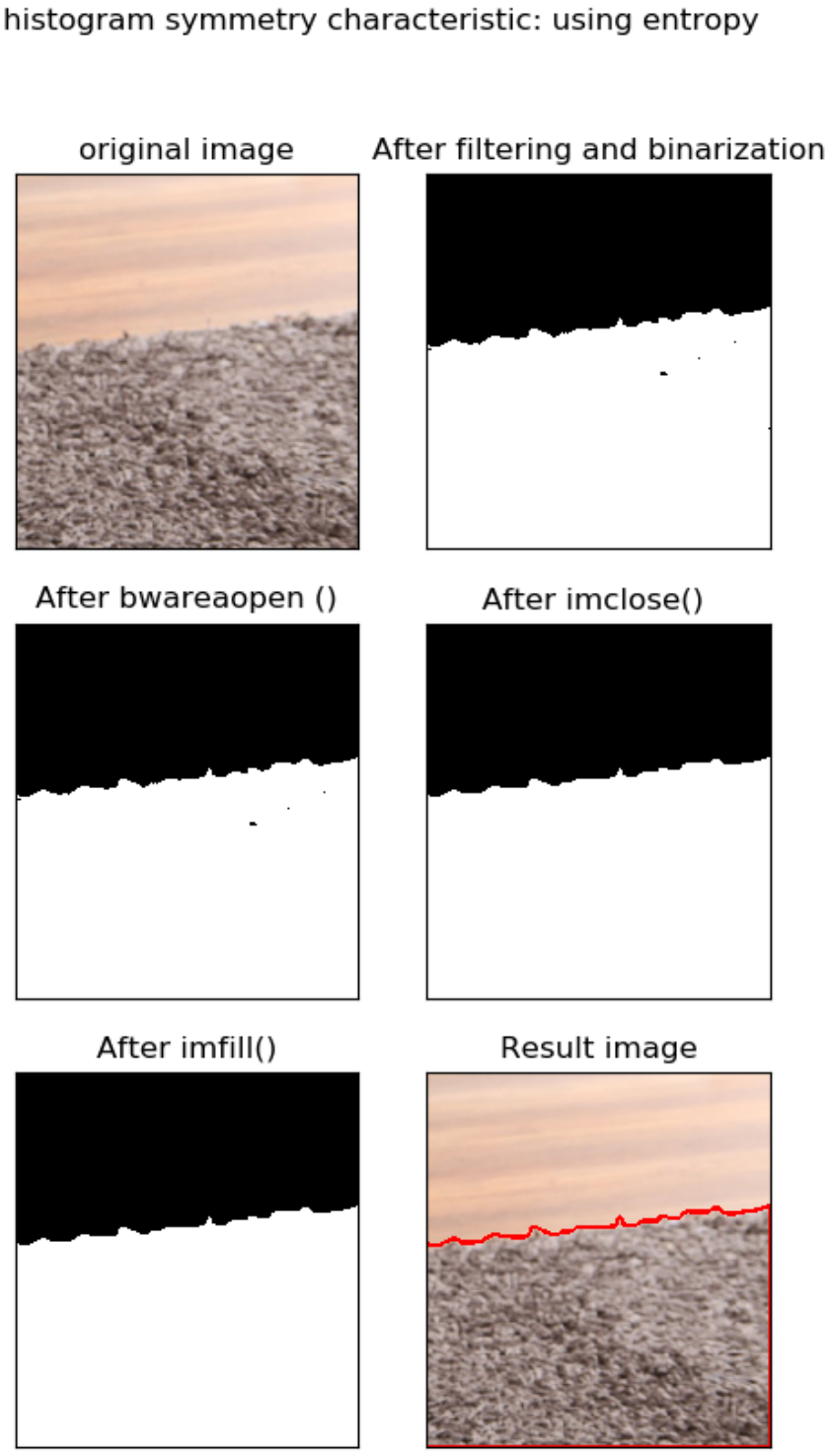

Fig. 3.28. Original image 1 segmentation result

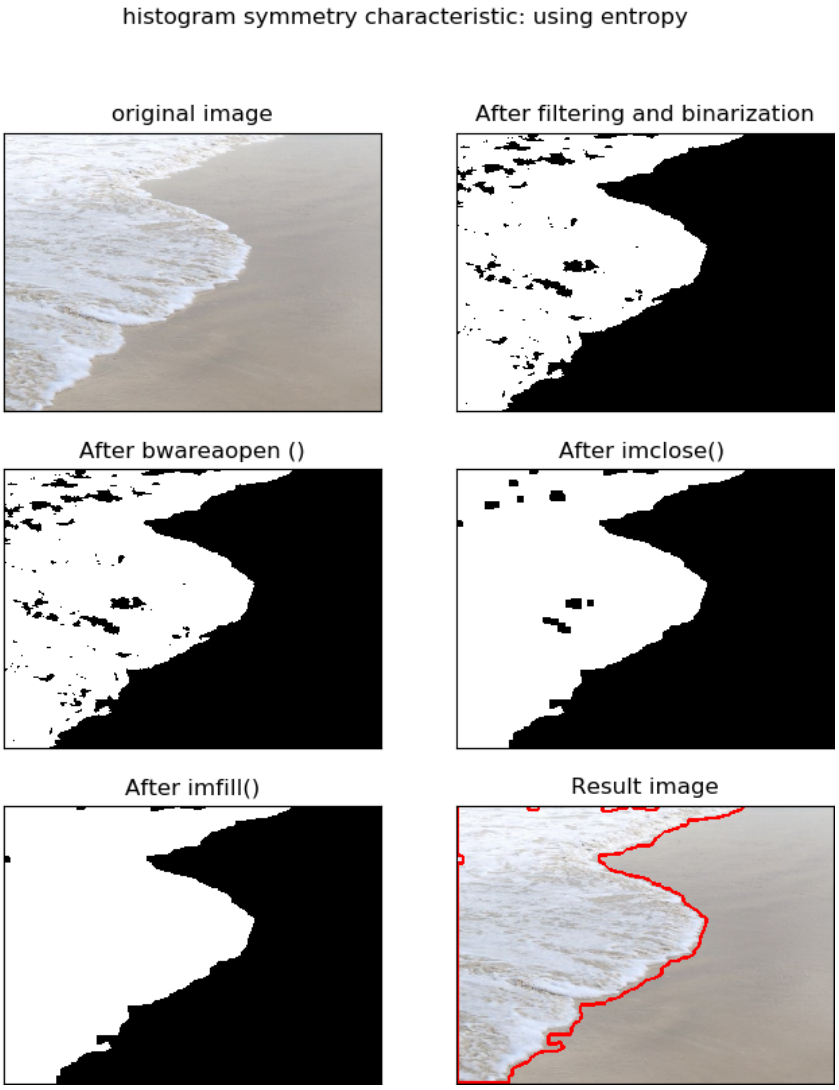

Fig. 3.29. Original image 2 segmentation result

4.b full source code

You can click ⋆here⋆ to see the full code for this part.

3. Comments

- ⋆ As can be seen from the processing results of the two selected original images, each statistical method has its own adaptation to the type of image.
- ⋆ From the two pictures of the experiment, the local entropy of the parameter picture has the best comprehensive performance.
- ⋆ In practical applications, you can try multiple times or combine several parameters to choose the most suitable texture segmentation method for the image.

4 Conclusion

In this exercise assignment, I learned the following application methods:

- ★ We can binarize the image using the following methods:
  1) thresholding
     a) upper thresholding
     b) lower thresholding
  2) double thresholding
     a) Otsu method
     b) Adaptive threshold method
- ★ For the segmentation of the skin part in the image,we should choose the formula of (3.5)-(3.7) to segment the skin part in the picture according to the characteristics (lighting conditions, race) of different pictures .
- ★ For the segmentation of color images, we usually operate in the CIE lab space, and for the classification of color pixel parts, we can use the following methods:
  a) Nearest neighbors method
  b) k-means method
- ★ For the segmentation of texture images, three main methods are used to describe texture: statistical, structural, and spectral.I have selected the following statistical parameters to perform. For statistical methods we can use the following parameters to describe the texture properties of the image:
  1) m - mean value of a random variable z
  2) s– Standard deviation of random variable z
  3) R - relative smoothness
  4) histogram symmetry characteristic : E– local entropy of the image

Note: The characteristics of each method are described in the previous comments and will not be repeated here

5 The answer to the questions

1) When is it appropriate to use Weber segmentation?

It can be seen from "Weber principle assumes that **the human eye does not perceive well the difference in gray levels between** $I(n)$ **and** $I(n)+W(I(n))$, where $W(I(n))$– Weber function" and from fig.3.8 that the image of the imageman after the use of Weber has almost no different differences, so Weber splitting is suitable for some image compression algorithms

2) What are the $\boldsymbol{a}$ and $\boldsymbol{b}$ color coordinates of the CIE Lab color space in a grayscale image?

The chromatic component of a color is given by two Cartesian coordinates.
a —– means the color position in the range from green (-128) to red (127)
$\boldsymbol{b}$ —– means the color position in the range blue (-128) to yellow (127).

3) What is the reason for performing an image segmentation in the CIE Lab color space and not in the original RGB one?

Because in the Lab color space, the value of lightness is separated from the value of the chromatic components of the color (hue, saturation),this can avoid the effect of brightness on color division.

I

Appendix

A Complete source code

§A.1 calculation

§A.2 1.1 upper and lower binarization thresholds

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import matplotlib.pyplot as plt

I = cv2.imread("sunflower.jpg",cv2.IMREAD_COLOR)

Igray = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
histsize=256
histrange=(0,256)
hist = cv2.calcHist([Igray],[0],None,[256],[0,255])
ret,Inew = cv2.threshold(Igray,0,255,cv2.THRESH_BINARY+ cv2.THRESH_OTSU
    )
ret,Inew1 = cv2.threshold (Igray,0,255,cv2.THRESH_BINARY_INV + cv2.
    THRESH_OTSU)
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
Inew = cv2.cvtColor(Inew, cv2.COLOR_BGR2RGB)
Inew1 = cv2.cvtColor(Inew1, cv2.COLOR_BGR2RGB)

#Display original image and histogram
plt.subplot(121),plt.title('original image'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Greyscale Image Histogram')
plt.plot(hist)
plt.show()
plt.waitforbuttonpress

#Display the image after the two methods are changed
plt.subplot(122),plt.title(' use lower threshold binarization.'),plt.
    axis ('off')
plt.imshow(Inew)
plt.subplot(121),plt.title(' use upper threshold binarization.'),plt.
    axis ('off')
plt.imshow(Inew1)
plt.show()
plt.waitforbuttonpress
```

§A.3 1.2 Double threshold binarization

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import matplotlib.pyplot as plt

I = cv2.imread("b1.jpg",cv2.IMREAD_COLOR)
Igray = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
histsize=256
histrange=(0,256)
hist = cv2.calcHist([Igray],[0],None,[256],[0,255])
Inew = cv2.adaptiveThreshold(Igray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY,11,2)
ret,Inew1 = cv2.threshold (Igray,0,255,cv2.THRESH_OTSU)
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
Inew = cv2.cvtColor(Inew, cv2.COLOR_BGR2RGB)
Inew1 = cv2.cvtColor(Inew1, cv2.COLOR_BGR2RGB)

#Display original image and histogram
plt.subplot(121),plt.title('original image'),plt.axis ('off')
plt.imshow(I)
plt.subplot(122),plt.title('Greyscale Image Histogram'),plt.axis ('off'
    )
plt.plot(hist)
plt.show()
plt.waitforbuttonpress

#Display the image after the two methods are changed
plt.subplot(122),plt.title('Adaptive methods'),plt.axis ('off')
plt.imshow(Inew)
plt.subplot(121),plt.title('OTSU'),plt.axis ('off')
plt.imshow(Inew1)
plt.show()
plt.waitforbuttonpress
```

§A.4 2.1 Images segmentation based on the Weber principle

and you can click ⋆here⋆ to return to reading the report

Problem 1

```
import cv2
import matplotlib.pyplot as plt
from numpy import zeros_like
import math
```

```

def Weber(I):
    if I>=0 and I<=88:
        W = 20 - 12*I/88
    elif I>88 and I<=138:
        W = 0.002*pow((I-88),2)
    else:
        W = 7*(I-138)/117 + 13
    return W

def Weber_segmentation(image):
    '''
    image : Input grayscale image
    '''
    m = image.shape[0]
    n = image.shape[1]
    W = zeros_like(image)
    Igraynew = zeros_like(image)
    for i in range(m):
        for j in range(n):
            W[i,j] = Weber(Igray[i,j])
    lower_bound = 0
    upper_bound = 0 + Weber(lower_bound)
    p = 0
    class1 = 1
    while p <m*n:
        print(p)
        for i in range(m):
            for j in range(n):
                if Igray[i,j]>=lower_bound and Igray[i,j]<=upper_bound:
                    Igraynew[i,j] = lower_bound
                    p = p+1
        lower_bound = math.floor(upper_bound + 1)
        upper_bound = lower_bound + Weber(lower_bound)
        class1 = class1 + 1
    return Igraynew,class1

########main################
#input the image and transform it into grayscale image
I = cv2.imread("face.jpg",cv2.IMREAD_COLOR)
I= cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
Igray = cv2.cvtColor(I,cv2.COLOR_RGB2GRAY)
#Weber segmentation
Igraynew,class1 = Weber_segmentation(Igray)
#display image
plt.subplot(121),plt.title('original grayscale image'),plt.xticks([]),
    plt.yticks([])
plt.imshow(Igray,cmap="gray")
plt.subplot(122),plt.title('Image after Weber segmentation'),plt.xticks
    ([]),plt.yticks([])
```

```
plt.imshow(Igraynew,cmap="gray")
plt.show()
plt.waitforbuttonpress
print(class1)
```

§A.5 2.2 Segmentation of RGB images by skin color

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import matplotlib.pyplot as plt
from numpy import zeros_like
# uniform day light illumination
def uniform_day_skin(image):
    '''
    image : RGB image
    '''
    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]
    m = image.shape[0]
    n = image.shape[1]
    skin = zeros_like(image)
    for i in range(m):
        for j in range(n):
            r = R[i,j]
            b = B[i,j]
            g = G[i,j]
            maxi = max(r,g,b)
            mini = min(r,g,b)
            if r>95 and g>40 and b>20 and maxi-mini>15 and abs(r-g)>15
               and r>g and r>b:
                skin[i,j] = 255
    return skin
#under flash light or daylight lateral illumination
def flash_light_skin(image):
    '''
    image : RGB image
    '''
    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]
    m = image.shape[0]
    n = image.shape[1]
    skin = zeros_like(image)
    for i in range(m):
        for j in range(n):
```

```
            r = R[i,j]
            b = B[i,j]
            g = G[i,j]
            if r>220 and g>210 and b>170 and  abs(r-g)<=15 and g>b and
               r>b:
                skin[i,j] = 255
    return skin
#using normalized RGB values
def normalized_RGB(image):
    '''
    image : RGB image
    '''
    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]
    m = image.shape[0]
    n = image.shape[1]
    skin = zeros_like(image)
    for i in range(m):
        for j in range(n):
            if (float(R[i,j])+float(G[i,j])+float(B[i,j])) != 0:
                r = float(R[i,j])/(float(R[i,j])+float(G[i,j])+float(B[
                   i,j]))
                b = float(B[i,j])/(float(R[i,j])+float(G[i,j])+float(B[
                   i,j]))
                g = float(G[i,j])/(float(R[i,j])+float(G[i,j])+float(B[
                   i,j]))
                if g != 0 and r/g>1.185 and r*b/pow((r+g+b),2) >0.107
                   and r*g/pow((r+g+b),2)>0.112:
                    skin[i,j] = 255
    return skin
#Uniform sunshine picture
I1 = cv2.imread("face3.jpg",cv2.IMREAD_COLOR)
I1= cv2.cvtColor(I1, cv2.COLOR_BGR2RGB)
#High exposure picture
I2 = cv2.imread("face13.jpg",cv2.IMREAD_COLOR)
I2= cv2.cvtColor(I2, cv2.COLOR_BGR2RGB)
# uniform day light illumination
uniform_1 = uniform_day_skin(I1)
uniform_2 = uniform_day_skin(I2)
#under flash light or daylight lateral illumination
flash_1 = flash_light_skin(I1)
flash_2 = flash_light_skin(I2)
#using normalized RGB values
normal_1 = normalized_RGB(I1)
normal_2 = normalized_RGB(I2)
#display image
plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
   ([])
plt.imshow(I1)
```

```
plt.subplot(222),plt.title('uniform day light formula'),plt.xticks([]),
    plt.yticks([])
plt.imshow(uniform_1,cmap="gray")
plt.subplot(223),plt.title('under flash light or daylight formula'),plt
    .xticks([]),plt.yticks([])
plt.imshow(flash_1,cmap="gray")
plt.subplot(224),plt.title('normalized RGB values formula'),plt.xticks
    ([]),plt.yticks([])
plt.imshow(normal_1,cmap="gray")
plt.show()
plt.waitforbuttonpress

plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(I2)
plt.subplot(222),plt.title('uniform day light formula'),plt.xticks([]),
    plt.yticks([])
plt.imshow(uniform_2,cmap="gray")
plt.subplot(223),plt.title('under flash light or daylight formula'),plt
    .xticks([]),plt.yticks([])
plt.imshow(flash_2,cmap="gray")
plt.subplot(224),plt.title('normalized RGB values formula'),plt.xticks
    ([]),plt.yticks([])
plt.imshow(normal_2,cmap="gray")
plt.show()
plt.waitforbuttonpress
```

§A.6 3.1 CIE Lab color space : Nearest neighbors method

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
from cv2 import sqrt
import matplotlib.pyplot as plt
from numpy import zeros_like
import numpy as np
def CIE_lab_nearest_segmentation(image,n_blocks):
    '''
    n_blocks : The category into which the image needs to be divided
    image : BGR image
    return : segmentedFrames,Iplot
    segmentedFrames : (RGB)The set of images after segmentation
    Iplot : (RGB)The distribution of segmented image on axis (a,b)
    '''
    Ilab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    Ilab = cv2.split(Ilab)
    #Color sample space generation
```

```
    sampleAreas = []
    def MouseHandler(event,x,y,flags,param):
        if event != cv2.EVENT_LBUTTONDBLCLK:
            return
        sampleAreas.append((x,y))
    cv2.imshow("image",image)
    cv2.setMouseCallback('image',MouseHandler)
    while len(sampleAreas) < n_blocks:
        cv2.waitKey(20)
    cv2.setMouseCallback('image',lambda*args:None)
    #Calculate average color
    colorMarksLAB = []
    colorMarksBGR = []
    for pix in sampleAreas:
        mask = zeros_like(Ilab[0])
        cv2.circle(mask,pix,10,255,-1)
        a = np.mean(Ilab[1][np.argwhere(mask > 0)])
        b = np.mean(Ilab[2][np.argwhere(mask > 0)])
        colorMarksLAB.append((a,b))
        colorMarksBGR.append(image[mask>0,:].mean(axis=(0)))
    #Calculates the minimum distance between all pixels and all colors
    distance = []
    for color in colorMarksLAB:
        distance.append(sqrt(pow((Ilab[1]-color[0]),2)+pow((Ilab[2]-
            color[1]),2)))
    distance_min = np.minimum.reduce(distance)
    #Calculates the color label for each pixel
    labels = zeros_like(Ilab[0],dtype= np.uint8)
    for i in range(len(colorMarksLAB)):
        mask = distance_min ==distance[i]
        labels[mask] = i
    #Split and store the original image
    segmentedFrames = []
    for i in range(len(colorMarksLAB)):
        Itmp = zeros_like(image)
        mask = labels == i
        Itmp[mask] = image[mask]
        segmentedFrames.append(Itmp)
    #Displays the color distribution in (a,b) coordinates
    Iplot = np.full((256,256,3),255,dtype=np.uint8)
    for i in range(len(colorMarksLAB)):
        Itmp = zeros_like(image)
        mask = labels == i
        Iplot[Ilab[1][mask],Ilab[2][mask],:] = colorMarksBGR[i]
    for i in range(n_blocks):
        segmentedFrames[i] = cv2.cvtColor(segmentedFrames[i],cv2.
            COLOR_BGR2RGB)
    Iplot = cv2.cvtColor(Iplot,cv2.COLOR_BGR2RGB)
    return segmentedFrames,Iplot

```

```
#input the image
#2 categories
I1 = cv2.imread("ball1.jpg",cv2.IMREAD_COLOR)
segmentedFrames1,Iplot1 = CIE_lab_nearest_segmentation(I1,2)
#3 categories
I2 = cv2.imread("ball4.webp",cv2.IMREAD_COLOR)
segmentedFrames2,Iplot2 = CIE_lab_nearest_segmentation(I2,3)
I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2RGB)
I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2RGB)

plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(I1)
plt.subplot(222),plt.title('segmented image 1'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[0])
plt.subplot(223),plt.title('segmented image 2'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[1])
plt.show()
plt.waitforbuttonpress

plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(I2)
plt.subplot(222),plt.title('segmented image 1'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[0])
plt.subplot(223),plt.title('segmented image 2'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[1])
plt.subplot(224),plt.title('segmented image 3'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[2])
plt.show()
plt.waitforbuttonpress
```

§A.7 3.2 CIE Lab color space : k-means method

and you can click ⋆here⋆ to return to reading the report

Python

```
import cv2
import matplotlib.pyplot as plt
from numpy import zeros_like
import numpy as np
def CIE_lab_kmeans_segmentation(image,n_blocks):
    '''
```

```
    n_blocks : The category into which the image needs to be divided
    image : BGR image
    return : segmentedFrames
    segmentedFrames : (RGB)The set of images after segmentation
    '''
    Ilab = cv2.cvtColor(image,cv2.COLOR_BGR2LAB)
    Ilab = cv2.split(Ilab)
    ab = cv2.merge([Ilab[1],Ilab[2]])
    ab = ab.reshape(-1,2).astype(np.float32)###
    #criteria is defined as not more than 10
    #iterations of difference between steps less than 1.
    criteria = (cv2.TERM_CRITERIA_EPS +cv2.TERM_CRITERIA_MAX_ITER
       ,10,1.0)
    ret,labels,centers = cv2.kmeans(ab,n_blocks,None,criteria,10,cv2.
       KMEANS_RANDOM_CENTERS)
    labels = labels.reshape((Ilab[0].shape))
    segmentedFrames = []
    for i in range(n_blocks):
        Itmp = zeros_like(image)
        mask = labels == i
        Itmp[mask] = image[mask,:]
        Itmp = cv2.cvtColor(Itmp,cv2.COLOR_BGR2RGB)
        segmentedFrames.append(Itmp)
    return segmentedFrames


#input the image
#2 categories
I1 = cv2.imread("ball1.jpg",cv2.IMREAD_COLOR)
segmentedFrames1 = CIE_lab_kmeans_segmentation (I1,2)
#3 categories
I2 = cv2.imread("ball4.webp",cv2.IMREAD_COLOR)
segmentedFrames2 = CIE_lab_kmeans_segmentation (I2,3)
I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2RGB)
I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2RGB)

plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
   ([])
plt.imshow(I1)
plt.subplot(222),plt.title('segmented image 1'),plt.xticks([]),plt.
   yticks([])
plt.imshow(segmentedFrames1[0])
plt.subplot(223),plt.title('segmented image 2'),plt.xticks([]),plt.
   yticks([])
plt.imshow(segmentedFrames1[1])
plt.show()
plt.waitforbuttonpress

plt.subplot(221),plt.title('original image'),plt.xticks([]),plt.yticks
   ([])
```

```
plt.imshow(I2)
plt.subplot(222),plt.title('segmented image 1'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[0])
plt.subplot(223),plt.title('segmented image 2'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[1])
plt.subplot(224),plt.title('segmented image 3'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames2[2])
plt.show()
plt.waitforbuttonpress
```

§A.8 4.1 Image texture segmentation: mean m as texture feature value

and you can click ⋆here⋆ to return to reading the report

Python

```
from ctypes import sizeof
from turtle import st
import cv2
from cv2 import sqrt
import matplotlib.pyplot as plt
from numpy import zeros_like
import numpy as np
from skimage import morphology,filters
def bwareaopen(A,dim,conn= 8):
    if A.ndim != 2 or A.dtype != np.uint8:
        return None
    # Find all connected components
    num,labels,stats,centers = cv2.connectedComponentsWithStats(A,
       connectivity=conn)
    #check siz of all aonnected components
    for i in range(num):
        if stats[i,cv2.CC_STAT_AREA] <dim:
            A[labels == i] = 0
    return A
def imfillholes(I):
    if I.ndim != 2 or I.dtype !=np.uint8:
        return None
    rows,cols = I.shape[0:2]
    mask = I.copy()
    #Fill mask from all horizontal borders
    for i in range(cols):
        if mask[0,i] == 0:
            cv2.floodFill(mask,None,(i,0),255,10,10)
```

```
28          if mask[rows-1,i] == 0:
29              cv2.floodFill(mask,None,(i,rows-1),255,10,10)
30      #Fill mask from all vertical borders
31      for i in range(rows):
32          if mask[i,0] == 0:
33              cv2.floodFill(mask,None,(0,i),255,10,10)
34          if mask[i,cols-1] == 0:
35              cv2.floodFill(mask,None,(cols-1,i),255,10,10)
36      #use the mask to create a resulting image
37      res = I.copy()
38      res[mask == 0] = 255
39      return res
40
41  def textures_segmentation_mean(image):
42      '''
43      image :(RGB)original image
44      '''
45      segmentedFrames = []
46      ###############binaryzation##########
47      Igray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
48      #Generate a 9x9 rectangular kernel
49      kernel = morphology.square(9)
50      #Calculate the local smoothness R of the image
51      Igray = Igray / 255.0
52      Igray_mean = cv2.blur(Igray,(9,9))
53      Igray_mean = Igray_mean *255
54      #Normalize the obtained float32 value
55      #and convert it to the range of 0~255
56      Igray_mean = np.uint8(Igray_mean)
57      #Binarize the image using
58      #the Otsu thresholding method
59      ret,BW1 = cv2.threshold(Igray_mean,0,255,cv2.THRESH_OTSU)
60      segmentedFrames.append(BW1)
61      ###############morphological filtering##########
62      #1. remove connected areas containing
63      #less than a given number of pixels
64      BW1 = bwareaopen(BW1,10000,conn= 8)
65      segmentedFrames.append(BW1)
66      #2. remove internal form defects or ««holes
67      nhood = cv2.getStructuringElement(cv2.MORPH_RECT,(9,9))
68      BW1 = cv2.morphologyEx(BW1,cv2.MORPH_CLOSE,nhood)
69      segmentedFrames.append(BW1)
70      #3. fill the remaining large ««holes
71      BW1 = imfillholes(BW1)
72      segmentedFrames.append(BW1)
73      contours,h = cv2.findContours(BW1,cv2.RETR_TREE,cv2.
            CHAIN_APPROX_NONE)
74      boundary = zeros_like(BW1)
75      cv2.drawContours(boundary,contours,-1,255,3)
76      segmentResults = image.copy()
```

```
77      segmentResults[boundary != 0,0] = 255
78      segmentResults[boundary != 0,1] = 0
79      segmentResults[boundary != 0,2] = 0
80      segmentedFrames.append(segmentResults)
81      return segmentedFrames
82
83  I1 = cv2.imread('sea1.jpg',cv2.IMREAD_COLOR)
84  I1 = cv2.cvtColor(I1,cv2.COLOR_BGR2RGB)
85  segmentedFrames1 = textures_segmentation_mean(I1)
86
87  plt.subplot(321),plt.title('original image'),plt.xticks([]),plt.yticks
        ([])
88  plt.imshow(I1,cmap="gray")
89  plt.subplot(322),plt.title('After filtering and binarization'),plt.
        xticks([]),plt.yticks([])
90  plt.imshow(segmentedFrames1[0],cmap="gray")
91  plt.subplot(323),plt.title('After bwareaopen ()'),plt.xticks([]),plt.
        yticks([])
92  plt.imshow(segmentedFrames1[1],cmap="gray")
93  plt.subplot(324),plt.title('After imclose() '),plt.xticks([]),plt.
        yticks([])
94  plt.imshow(segmentedFrames1[2],cmap="gray")
95  plt.subplot(325),plt.title('After imfill()'),plt.xticks([]),plt.yticks
        ([])
96  plt.imshow(segmentedFrames1[3],cmap="gray")
97  plt.subplot(326),plt.title('Result image'),plt.xticks([]),plt.yticks
        ([])
98  plt.imshow(segmentedFrames1[4])
99  plt.suptitle('the features of smoothness:mean value')
100 plt.show()
101 plt.waitforbuttonpress
```

§A.9 4.2 Image texture segmentation: Standard deviation as texture feature value

and you can click ⋆here⋆ to return to reading the report

Python

```
1  from ctypes import sizeof
2  from turtle import st
3  import cv2
4  from cv2 import sqrt
5  import matplotlib.pyplot as plt
6  from numpy import zeros_like
7  import numpy as np
8  from skimage import morphology,filters
9  def bwareaopen(A,dim,conn= 8):
```

```
    if A.ndim != 2 or A.dtype != np.uint8:
        return None
    # Find all connected components
    num,labels,stats,centers = cv2.connectedComponentsWithStats(A,
       connectivity=conn)
    #check siz of all aonnected components
    for i in range(num):
        if stats[i,cv2.CC_STAT_AREA] <dim:
            A[labels == i] = 0
    return A
def imfillholes(I):
    if I.ndim != 2 or I.dtype !=np.uint8:
        return None
    rows,cols = I.shape[0:2]
    mask = I.copy()
    #Fill mask from all horizontal borders
    for i in range(cols):
        if mask[0,i] == 0:
            cv2.floodFill(mask,None,(i,0),255,10,10)
        if mask[rows-1,i] == 0:
            cv2.floodFill(mask,None,(i,rows-1),255,10,10)
    #Fill mask from all vertical borders
    for i in range(rows):
        if mask[i,0] == 0:
            cv2.floodFill(mask,None,(0,i),255,10,10)
        if mask[i,cols-1] == 0:
            cv2.floodFill(mask,None,(cols-1,i),255,10,10)
    #use the mask to create a resulting image
    res = I.copy()
    res[mask == 0] = 255
    return res

def textures_segmentation_standard_deviation(image):
    '''
    image :(RGB)original image
    '''
    segmentedFrames = []
    ###############binaryzation##########
    Igray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    #Generate a 9x9 rectangular kernel
    kernel = morphology.square(9)
    #Calculate the local smoothness R of the image
    Igray = Igray / 255.0
    Igray_2 = Igray ** 2
    Igray_2_mean = cv2.blur(Igray_2,(9,9))
    Igray_mean = cv2.blur(Igray,(9,9))
    Igray_mean_2 = Igray_mean ** 2
    local_variance = np.maximum((Igray_2_mean - Igray_mean_2),0)
    local_variance = local_variance *255*255
    local_standard_deviation = sqrt(local_variance)
```

```
    m,n = Igray.shape
    smooth_R = zeros_like(Igray)
    print(m,n)
    smooth_Rim = np.uint8(local_standard_deviation)
    #Binarize the image using
    #the Otsu thresholding method
    ret,BW1 = cv2.threshold(smooth_Rim,0,255,cv2.THRESH_OTSU)
    segmentedFrames.append(BW1)
    ###############morphological filtering##########
    #1. remove connected areas containing
    #less than a given number of pixels
    BW1 = bwareaopen(BW1,10000,conn= 8)
    segmentedFrames.append(BW1)
    #2. remove internal form defects or ««holes
    nhood = cv2.getStructuringElement(cv2.MORPH_RECT,(9,9))
    BW1 = cv2.morphologyEx(BW1,cv2.MORPH_CLOSE,nhood)
    segmentedFrames.append(BW1)
    #3. fill the remaining large ««holes
    BW1 = imfillholes(BW1)
    segmentedFrames.append(BW1)
    contours,h = cv2.findContours(BW1,cv2.RETR_TREE,cv2.
        CHAIN_APPROX_NONE)
    boundary = zeros_like(BW1)
    cv2.drawContours(boundary,contours,-1,255,3)
    segmentResults = image.copy()
    segmentResults[boundary != 0,0] = 255
    segmentResults[boundary != 0,1] = 0
    segmentResults[boundary != 0,2] = 0
    segmentedFrames.append(segmentResults)
    return segmentedFrames

I1 = cv2.imread('sea1.jpg',cv2.IMREAD_COLOR)
I1 = cv2.cvtColor(I1,cv2.COLOR_BGR2RGB)
segmentedFrames1 = textures_segmentation_standard_deviation(I1)

plt.subplot(321),plt.title('original image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(I1,cmap="gray")
plt.subplot(322),plt.title('After filtering and binarization'),plt.
    xticks([]),plt.yticks([])
plt.imshow(segmentedFrames1[0],cmap="gray")
plt.subplot(323),plt.title('After bwareaopen ()'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[1],cmap="gray")
plt.subplot(324),plt.title('After imclose() '),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[2],cmap="gray")
plt.subplot(325),plt.title('After imfill()'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(segmentedFrames1[3],cmap="gray")
```

```
plt.subplot(326),plt.title('Result image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(segmentedFrames1[4])
plt.suptitle('texture characteristic: standard deviation')
plt.show()
plt.waitforbuttonpress
```

§A.10 4.3 Image texture segmentation: Relative Smoothness R as texture feature value

and you can click ⋆here⋆ to return to reading the report

Python

```
from ctypes import sizeof
from turtle import st
import cv2
from cv2 import sqrt
import matplotlib.pyplot as plt
from numpy import zeros_like
import numpy as np
from skimage import morphology,filters
def bwareaopen(A,dim,conn= 8):
    if A.ndim != 2 or A.dtype != np.uint8:
        return None
    # Find all connected components
    num,labels,stats,centers = cv2.connectedComponentsWithStats(A,
       connectivity=conn)
    #check siz of all aonnected components
    for i in range(num):
        if stats[i,cv2.CC_STAT_AREA] <dim:
            A[labels == i] = 0
    return A
def imfillholes(I):
    if I.ndim != 2 or I.dtype !=np.uint8:
        return None
    rows,cols = I.shape[0:2]
    mask = I.copy()
    #Fill mask from all horizontal borders
    for i in range(cols):
        if mask[0,i] == 0:
            cv2.floodFill(mask,None,(i,0),255,10,10)
        if mask[rows-1,i] == 0:
            cv2.floodFill(mask,None,(i,rows-1),255,10,10)
    #Fill mask from all vertical borders
    for i in range(rows):
        if mask[i,0] == 0:
            cv2.floodFill(mask,None,(0,i),255,10,10)
```

```
34          if mask[i,cols-1] == 0:
35              cv2.floodFill(mask,None,(cols-1,i),255,10,10)
36      #use the mask to create a resulting image
37      res = I.copy()
38      res[mask == 0] = 255
39      return res
40
41  def textures_segmentation_smoothness(image):
42      '''
43      image :(RGB)original image
44      '''
45      segmentedFrames = []
46      ###############binaryzation##########
47      Igray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
48      #Generate a 9x9 rectangular kernel
49      kernel = morphology.square(9)
50      #Calculate the local smoothness R of the image
51      Igray = Igray / 255.0
52      Igray_2 = Igray ** 2
53      Igray_2_mean = cv2.blur(Igray_2,(9,9))
54      Igray_mean = cv2.blur(Igray,(9,9))
55      Igray_mean_2 = Igray_mean ** 2
56      local_variance = np.maximum((Igray_2_mean - Igray_mean_2),0)
57      local_variance = local_variance *255*255
58      m,n = Igray.shape
59      smooth_R = zeros_like(Igray)
60      print(m,n)
61      for i in range(m):
62          for j in range(n):
63              smooth_R[i,j] = 1 - 1/(1+local_variance[i,j])
64      #Normalize the obtained float32 value
65      #and convert it to the range of 0~255
66      smooth_Rim = np.uint8(smooth_R*255)
67      #Binarize the image using
68      #the Otsu thresholding method
69      ret,BW1 = cv2.threshold(smooth_Rim,0,255,cv2.THRESH_OTSU)
70      segmentedFrames.append(BW1)
71      ###############morphological filtering##########
72      #1. remove connected areas containing
73      #less than a given number of pixels
74      BW1 = bwareaopen(BW1,10000,conn= 8)
75      segmentedFrames.append(BW1)
76      #2. remove internal form defects or ««holes
77      nhood = cv2.getStructuringElement(cv2.MORPH_RECT,(9,9))
78      BW1 = cv2.morphologyEx(BW1,cv2.MORPH_CLOSE,nhood)
79      segmentedFrames.append(BW1)
80      #3. fill the remaining large ««holes
81      BW1 = imfillholes(BW1)
82      segmentedFrames.append(BW1)
83      contours,h = cv2.findContours(BW1,cv2.RETR_TREE,cv2.
```

```
        CHAIN_APPROX_NONE)
    boundary = zeros_like(BW1)
    cv2.drawContours(boundary,contours,-1,255,3)
    segmentResults = image.copy()
    segmentResults[boundary != 0,0] = 255
    segmentResults[boundary != 0,1] = 0
    segmentResults[boundary != 0,2] = 0
    segmentedFrames.append(segmentResults)
    return segmentedFrames

I1 = cv2.imread('sea1.jpg',cv2.IMREAD_COLOR)
I1 = cv2.cvtColor(I1,cv2.COLOR_BGR2RGB)
segmentedFrames1 = textures_segmentation_smoothness(I1)


plt.subplot(321),plt.title('original image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(I1,cmap="gray")
plt.subplot(322),plt.title('After filtering and binarization'),plt.
    xticks([]),plt.yticks([])
plt.imshow(segmentedFrames1[0],cmap="gray")
plt.subplot(323),plt.title('After bwareaopen ()'),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[1],cmap="gray")
plt.subplot(324),plt.title('After imclose() '),plt.xticks([]),plt.
    yticks([])
plt.imshow(segmentedFrames1[2],cmap="gray")
plt.subplot(325),plt.title('After imfill()'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(segmentedFrames1[3],cmap="gray")
plt.subplot(326),plt.title('Result image'),plt.xticks([]),plt.yticks
    ([])
plt.imshow(segmentedFrames1[4])
plt.suptitle('the features of smoothness: relative smoothness R')
plt.show()
plt.waitforbuttonpress
```

§A.11 4.4 Image texture segmentation: Local Entropy E as texture feature value

and you can click ⋆here⋆ to return to reading the report

Python

```
from turtle import st
import cv2
from cv2 import sqrt
import matplotlib.pyplot as plt
```

```
from numpy import zeros_like
import numpy as np
from skimage import morphology,filters
def bwareaopen(A,dim,conn= 8):
    if A.ndim != 2 or A.dtype != np.uint8:
        return None
    # Find all connected components
    num,labels,stats,centers = cv2.connectedComponentsWithStats(A,
       connectivity=conn)
    #check siz of all aonnected components
    for i in range(num):
        if stats[i,cv2.CC_STAT_AREA] <dim:
            A[labels == i] = 0
    return A
def imfillholes(I):
    if I.ndim != 2 or I.dtype !=np.uint8:
        return None
    rows,cols = I.shape[0:2]
    mask = I.copy()
    #Fill mask from all horizontal borders
    for i in range(cols):
        if mask[0,i] == 0:
            cv2.floodFill(mask,None,(i,0),255,10,10)
        if mask[rows-1,i] == 0:
            cv2.floodFill(mask,None,(i,rows-1),255,10,10)
    #Fill mask from all vertical borders
    for i in range(rows):
        if mask[i,0] == 0:
            cv2.floodFill(mask,None,(0,i),255,10,10)
        if mask[i,cols-1] == 0:
            cv2.floodFill(mask,None,(cols-1,i),255,10,10)
    #use the mask to create a resulting image
    res = I.copy()
    res[mask == 0] = 255
    return res

def textures_segmentation_entropy(image):
    '''
    image :(RGB)original image
    '''
    segmentedFrames = []
    ################binaryzation##########
    Igray = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    #Generate a 9x9 rectangular kernel
    kernel = morphology.square(9)
    #Calculate the entropy of the picture
    E = filters.rank.entropy(Igray,kernel).astype(np.float32)
    #Normalize the obtained float32 value
    #and convert it to the range of 0~255
    Eim = (E-E.min()) / (E.max() - E.min())
```

```
    Eim = np.uint8(Eim*255)
    #Binarize the image using
    #the Otsu thresholding method
    ret,BW1 = cv2.threshold(Eim,0,255,cv2.THRESH_OTSU)
    segmentedFrames.append(BW1)
    ###############morphological filtering##########
    #1. remove connected areas containing
    #less than a given number of pixels
    BW1 = bwareaopen(BW1,10000,conn= 8)
    segmentedFrames.append(BW1)
    #2. remove internal form defects or ««holes
    nhood = cv2.getStructuringElement(cv2.MORPH_RECT,(9,9))
    BW1 = cv2.morphologyEx(BW1,cv2.MORPH_CLOSE,nhood)
    segmentedFrames.append(BW1)
    #3. fill the remaining large ««holes
    BW1 = imfillholes(BW1)
    segmentedFrames.append(BW1)
    contours,h = cv2.findContours(BW1,cv2.RETR_TREE,cv2.
       CHAIN_APPROX_NONE)
    boundary = zeros_like(BW1)
    cv2.drawContours(boundary,contours,-1,255,3)
    segmentResults = image.copy()
    segmentResults[boundary != 0,0] = 255
    segmentResults[boundary != 0,1] = 0
    segmentResults[boundary != 0,2] = 0
    segmentedFrames.append(segmentResults)
    return segmentedFrames

I1 = cv2.imread('sea1.jpg',cv2.IMREAD_COLOR)
I1 = cv2.cvtColor(I1,cv2.COLOR_BGR2RGB)
segmentedFrames1 = textures_segmentation_entropy(I1)


plt.subplot(321),plt.title('original image'),plt.xticks([]),plt.yticks
   ([])
plt.imshow(I1,cmap="gray")
plt.subplot(322),plt.title('After filtering and binarization'),plt.
   xticks([]),plt.yticks([])
plt.imshow(segmentedFrames1[0],cmap="gray")
plt.subplot(323),plt.title('After bwareaopen ()'),plt.xticks([]),plt.
   yticks([])
plt.imshow(segmentedFrames1[1],cmap="gray")
plt.subplot(324),plt.title('After imclose() '),plt.xticks([]),plt.
   yticks([])
plt.imshow(segmentedFrames1[2],cmap="gray")
plt.subplot(325),plt.title('After imfill()'),plt.xticks([]),plt.yticks
   ([])
plt.imshow(segmentedFrames1[3],cmap="gray")
plt.subplot(326),plt.title('Result image'),plt.xticks([]),plt.yticks
   ([])
```

```
97  plt.imshow(segmentedFrames1[4])
98  plt.suptitle('histogram symmetry characteristic: using entropy')
99  plt.show()
100 plt.waitforbuttonpress
```