HOMEWORK ASSIGNMENT #3

Morphological Processing, Texture Analysis

Poy Lu 呂栢頤
D09944015
網媒所博一
ariapoy@gmail.com

April 21, 2021

1 Problem 1: MORPHOLOGICAL PROCESSING

A binary image, **sample1.png**, is given in Figure 1. Please implement several morphological operations and provide discussions about the results. (Note that the white pixels represent foreground objects and the black pixels are background.)

Given an image, **sample1.jpg**.

Original image **sample1.jpg** for question Problem 1: MORPHOLOGICAL PROCESSING.

1.1 (a)

Perform boundary extraction on **sample1.png** to extract the objects'boundaries and output the result as **result1.png**.

Motivation Extract the boundary (or outline) of an object. Recall of generalized **dilation** and **erosion** in *Lec 4 page 32*.

- dilation: $G(j,k) = F(j,k) \oplus H(j,k)$
- erosion: $G(j,k) = F(j,k) \ominus H(j,k)$

where $H(j,k)$ is **structure element**.

Figure 1: **sample1.jpg**

Approach We could follow the formula in *Lec 4 page 39*. The formula of **boundary extraction** is

$$\beta(F(j,k)) = F(j,k) - (F(j,k) \ominus H(j,k))$$

It is the original image **minus** its erosion.

I start from generalized **dilation** and **erosion**. This is my **core function** for (a), (b) & (c).

1. Convolution with **structure element** $H(j,k)$ to get $T_{r,c}\{F(j,k)\}$.
2. Conduct **erosion** or **dilation**:
   - **erosion**: $G(j,k) = \cap_{(r,c)\in H} T_{r,c}\{F(j,k)\}$
     As we **intersection** of all $T_{r,c}$, we could check the **sum of matching** $T_{r,c}$ is **equal** of # 1 in $H_{j,k}$.
   - **dilation**: $G(j,k) = \cup_{(r,c)\in H} T_{r,c}\{F(j,k)\}$
     As we **union** of all $T_{r,c}$, we could check the **sum of matching** $T_{r,c}$ is **greater than** 0.

This approach is transform **intersection** & **union** as **convolution with condition**.

Then conduct the **boundary extraction** by formula.

Figure 2: **result1.jpg** Boundary extraction

1. Design the **structure element** $H(j,k)$ as well as **origin points**.
2. Calculate **erosion** $F(j,k) \ominus H(j,k)$
3. Obtain **boundary extraction**.

Performance of results In the end, I choose same **structure element** in *Lec 4 page 39*, that is

$$H(j,k) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

with **center point as origin**.

Result of problem 1(a): **result1.jpg** Boundary extraction.

Discussion I try some different **structure element**.

Different of structure element:

$$H(j,k) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

TBA...

$$H(j,k) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

TBA...

Different size of structure element: Size $H_{5\times5}(j,k)$ **TBA**...

1.2 (b)

Perform hole filling on **sample1.png** and output the result as **result2.png**.

Motivation Given a pixel inside a boundary, hole filling attempts to fill that boundary with object pixels.

First, this operation uses **dilation**($\oplus$). We could reuse the core function designed in (a). But I find out the result is not good as my expectation.

Second, as we could treat it as *Fill tool in Painter*, we could use flood fill algorithm for this problem.

Approach My idea is start from **breadth-first search (BFS)**. The idea comes from LeedCode discussion.

1. Initialize empty queue and add the **start point** to fill.
2. Scan the color of position (i, j)
   - if the current is not visited &
   - the color of current is the same as queue of the start point.

   We fill the new color of this pixels.

I use the trick that start from the **top-left corner pixel** with **black**. And fill up all **background** as **tag** (2). Then I keep it as background and replace other pixels with **white color** (255).

Performance of results In the end, I choose the **flood fill** algorithm.

Result of problem 1(b): **result2.jpg** Hole filling with flood fill.

Discussion We could follow the formula in *Lec 4 page 41*. The formula of **hole filling** is

$$G(j,k) = ((G_{i-1}(j,k) \oplus H(j,k)) \cap F^c(j,k)) \cup F(j,k)$$

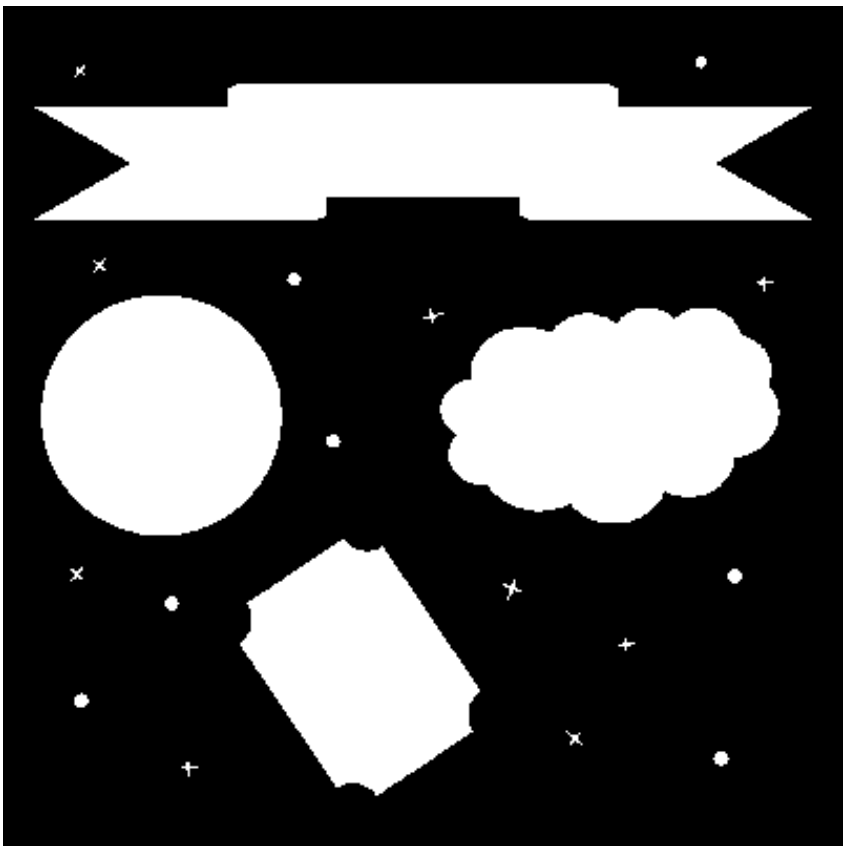

Figure 3: **result2.jpg** Hole filling with flood fill

where $i = 1, 2, \ldots$, $F^c(j,k)$ means the **complementary** of original image. It is the **interior** of original image **union** with **boundary**.

My approach is

1. Calculate complementary $F^c(j,k)$
2. Dilation then **intersection** with complementary $(F(j,k) \oplus H(j,k)) * F^c(j,k)$
   As we just get $\{0,1\}$ of dilation as well as complementary results, we could times $(*)$ them as **intersection**.

Moreover, I add parameters of # **repeated** of hole filling operations.

Give the same **structure element**

$$H(j,k) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

with **center point as origin**. And repeat hole filling with **12** times.

Here is the result: **result2_lec4.jpg** Hole filling in Lec 4 page 41

Although **result2.jpg** Hole filling with flood fill shows we successfully fill all holes, it expands original objects and connect to each other. Why does it occur? I consider that

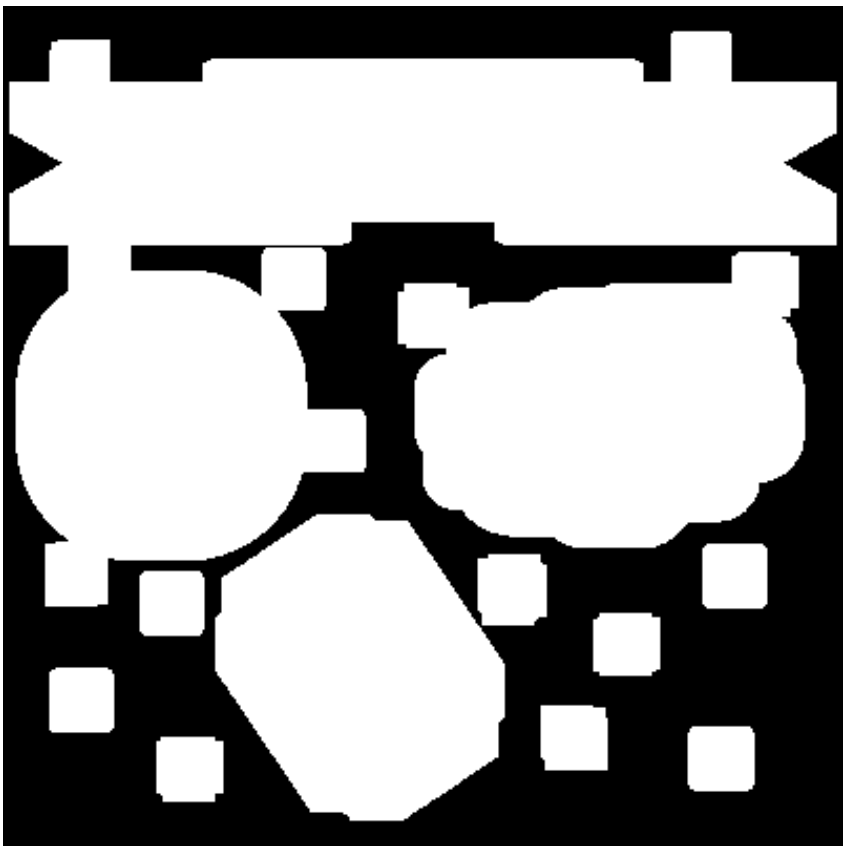

Figure 4: **result2_lec4.jpg** Hole filling in Lec 4 page 41

my approach is not as good as *Lec 4 page 41*. As I don't choose the **interior points** in the objects. So all objects become larger and larger.

1.3 (c)

Please design an algorithm to count the number of objects in Figure 1. Describe the steps in detail and specify the corresponding parameters.

Motivation Scan an image and groups its pixels into components based on pixel connectivity.

First, this operation uses **dilation**($\oplus$) again. We could reuse the core function designed in (a). However, I can't count the number of objects as I don't calculate it **iteratively**.

Second, I use **two-stage scan** for couting the number of objects. The detail steps is shown below.

Approach My approach is

1. Initialize the object with tag -1.

Figure 5: # 20 Objects with color

2. Scan the 8-neighbors of object $F(j,k) \neq 0$ as `G_subarray`.
   - if there are two above tags in `G_subarray`, we store it as **connected cluster**.
   - else substitute the new cluster tag for -1 or propogate existed tag in `G_subarray`.
3. Get `label_object_adj_mat` from **connected cluster**.
4. Check the connectivity of label object adjacency matrix by A^k.
   where k is number of clusters (candidate label objects).
5. Merge the connect clusters as final label objects. And count it!

Performance of results I use the **result2.jpg** Hole filling with flood fill than count it!

Result of problem 1(c): # 20 Objects with color with 20 objects.

Discussion We could also conduct it for **result2_lec4.jpg** Hole filling in Lec 4 page 41. Result of problem 1(c): # 20 Objects with color with 11 objects.

Figure 6: # 11 Objects with color (Lec 4)