HW 7

b05902019 資工三 蔡青邑

`Using "lena.bmp" as input image.`

Python Packages I used

- `skimage.io`: for basic image i/o.
- `numpy`: for the convience of array manipulation.
- `tqdm`: for visualizing the progress of my while loop.

Some Other Functions I Build

- **binarize(img, lower_expand, upper_expand, threshold)**: binarize the image(img) according to the threshold and return it.
- **yokoi(f)**: given the four type in `f` (i.e. `['r', 'r', 'r', 'r']`), returning the corresponding yokoi number.

How I Implemented Thinning Operator

> *Following the instruction of our slides, the whole algorithm is divided into three parts.*

The first part: Yokoi Number

1. After reading the input image, I binarized it.
2. Then followed by the requirement, I shrink the image into 64 x 64.

```
height = int(height / 8)
width = int(width / 8)
for i in range(height + 2):
    frame.append([])
    for j in range(width + 2):
        frame[-1].append(0)
```

3. Then I ouput the yokoi number according to the formulas and definition of our slide with my self-defined dunction, `yokoi()`.

$$h(b,c,d,e) = \begin{cases} q & if\ b = c\ and\ (d \neq b \vee e \neq b) \\ r & if\ b = c\ and\ (d = b \wedge e = b) \\ s & if\ b \neq c \end{cases}$$

$$f(a_1,a_2,a_3,a_4) = \begin{cases} 5 & if\ a_1 = a_2 = a_3 = a_4 = r \\ n & \text{where } n = \text{ number of } \{a_k | a_k = q\}, othe \end{cases}$$

```
def yokoi_operator(shrink_lena, height, width):
    frame = []
    for i in range(height + 2):
        frame.append([])
        for j in range(width + 2):
            frame[-1].append(0)
    # print(np.array(frame).shape)
    # put in a 0, 0 frame
    for i in range(height):
        for j in range(width):
            frame[i + 1][j + 1] = shrink_lena[i][j]
    ans = [[]]
    delta_c = np.array([[1, 0], [0, 1], [-1, 0], [0, -1]])
    delta_d = np.array([[1, 1], [-1, 1], [-1, -1], [1, -1]])
    delta_e = np.array([[0, 1], [-1, 0], [0, -1], [1, 0]])
    after_yokoi = []
    for i in range(1, height + 2 - 1):
        temp = []
```

```
        for j in range(1, width + 2 - 1):
            if(frame[i][j] == 0):
                # print(' ', end = '')
                temp.append(0)
            else:
                f = []
                for d in range(4):
                    Type = 'chiu'
                    o = np.array([i, j])
                    b, c, d, e = o, o + delta_c[d] , o + delta_d[d],
o + delta_e[d]
                    # print(b, c, d, e)
                    b, c, d, e = frame[b[0]][b[1]], frame[c[0]]
[c[1]], frame[d[0]][d[1]], frame[e[0]][e[1]]
                    if(b == c and (d != b or e != b)):
                        Type = 'q'
                    elif(b == c):
                        Type = 'r'
                    else:
                        Type = 's'
                    f.append(Type)
                ans = yokoi(f)
                temp.append(ans)
                # print(' ' if(ans == 0) else ans, end = '')
        # print()
        after_yokoi.append(temp)
    return after_yokoi
```

The second part: Pair Relationship Operator

According to the slide, I run the following functions on each pixel. (For those pixels on the border of the image, I chose to deem all the values outside the image as zeros.)

$$H \text{ function}: m = \text{" 1 "}, \text{ means " edge " in } \mathbb{Y}\text{o}\mathbb{k}\text{oi}.$$

$$h(a,m) = \begin{cases} 1 & ,if\ a = m \\ 0 & ,\text{otherwise} \end{cases}$$

$$y = \begin{cases} q & if\ \Sigma_{n=1}^{4} h(x_n, m) < 1 \text{ or } x_0 \neq m \\ p & if\ \Sigma_{n=1}^{4} h(x_n, m) \geq 1 \text{ and } x_0 = m \end{cases}$$

```
def pair_relationship(yokoi_img, height, width):
    frame = np.zeros((height + 2, width + 2), dtype = int)
    ans = np.zeros((height, width), dtype = object)
    for i in range(height):
        for j in range(width):
            frame[i + 1][j + 1] = yokoi_img[i][j]
    for i in range(height):
        for j in range(width):
            if(frame[i + 1][j + 1] == 1):
                if(frame[i + 1 + 1][j + 1] == 1 or
                  frame[i + 1][j + 1 + 1] == 1 or
                  frame[i + 1 - 1][j + 1] == 1 or
                  frame[i + 1][j + 1 - 1] == 1):
                    ans[i][j] = 'p'
            elif(frame[i + 1][j + 1] == 0):
                ans[i][j] = 0
            else:
                ans[i][j] = 'q'
    # print(frame)
    return ans
```

The third part: Connected Shrink Operator

By the same token, according to the slide, I run the following functions on each pixel. (For those pixels on the border of the image, I chose to deem all the values outside the image as zeros.)

$$H \text{ function : yokoi corner } \Rightarrow \text{“q”}$$

$$h(b,c,d,e) = \begin{cases} 1 & if\ b = c\ and\ (d \neq b \vee e \neq b) \\ 0 & \text{, otherwise} \end{cases}$$

$$\mathbb{O}\text{utput : } f(a_1,a_2,a_3,a_4) = \begin{cases} g & if \text{ exactly one of } a_n = 1, n = 1 \sim \\ x & \text{, otherwise} \end{cases}$$

```
def shrinking_operator(paired_img, shrink_lena, height, width):
    frame = np.zeros((66, 66), dtype = object)
    ans = np.zeros((64, 64), dtype = int)
    for i in range(height):
        for j in range(width):
            frame[i + 1][j + 1] = shrink_lena[i][j]
    for i in range(height):
        for j in range(width):
            if(paired_img[i][j] == 'p' and 1 == frame[i + 1][j + 1]):
                a = (1 == frame[i + 1 + 0][j + 1 + 1] and (frame[i +
1 - 1][j + 1 + 1] != 1 or frame[i + 1 - 1][j + 1 + 0] != 1))
                b = (1 == frame[i + 1 - 1][j + 1 + 0] and (frame[i +
1 - 1][j + 1 - 1] != 1 or frame[i + 1 + 0][j + 1 - 1] != 1))
                c = (1 == frame[i + 1 + 0][j + 1 - 1] and (frame[i +
1 + 1][j + 1 - 1] != 1 or frame[i + 1 + 1][j + 1 + 0] != 1))
                d = (1 == frame[i + 1 + 1][j + 1 + 0] and (frame[i +
1 + 1][j + 1 + 1] != 1 or frame[i + 1 + 0][j + 1 + 1] != 1))
                if(int(a) + int(b) + int(c) + int(d) == 1):
                    frame[i + 1][j + 1] = 0
    for i in range(height):
        for j in range(width):
            shrink_lena[i][j]= frame[i + 1][j + 1]
    return shrink_lena
```

My main function and result:

```
from tqdm import tqdm_notebook as tqdm
import numpy as np
height = len(shrink_lena)
width = len(shrink_lena[0])

last_shrink_lena = [[0] * 64] * 64
pbar = tqdm(total=0)
while(1):
    pbar.update(1)

    for i in range(height):
        for j in range(width):
            last_shrink_lena[i][j] = shrink_lena[i][j]

    after_yokoi = yokoi_operator(shrink_lena, height, width)
    paired = pair_relationship(after_yokoi, height, width)
    shrink_lena = shrinking_operator(paired, shrink_lena, height,
width)

    output = np.asarray(shrink_lena, dtype = int) * 255
    last_output = io.imread("thinning.png")
    if(np.array_equal(output, last_output)):
        break
```

```
    else:
        io.imsave("thinning.png", output)
pbar.close()
```