Computer-vision-Homework 4

Mathematical Morphology – Binary Morphology

Due date：2 Nov 2021

Programming language: python 3.9.7

Import lib:

- OpenCv: to read and write the image file
- Numpy: to work with the arrays

Original image: lena.bmp

[512(width),512(height),1channel(cv2.IMREAD_GRAYSCALE)]

Code explanation:

```
lena = cv2.imread('lena.bmp', cv2.IMREAD_GRAYSCALE)
```

- imread to load a file with cv2.IMREAD_GRAYSCALE(or using 0 as the parameter of the function)

```
def binary(image):
    lena_binary = np.zeros(image.shape, np.uint8)
    rows, columns = lena_binary.shape
    for i in range(rows):
        for j in range(columns):
            if image[i][j] >= 128:
                lena_binary[i][j] = 255
            else:
                lena_binary[i][j] = 0
    return lena_binary
```

- Binarize lena.bmp with threshold(0~127, 128~255)

Q1：Dilation on a binary image

```
def dilation(image, kernel):
    dilimage = np.zeros(image.shape, np.uint8)
    m, n = image.shape
    for i in range(m):
        for j in range(n):
            if image[i][j] > 0:
                for position in kernel:
                    p, q = position
                    if 0 <= (i + p) <= (m - 1) and 0 <= (j + q) <= (n - 1):
                        dilimage[i + p][j + q] = 255
    return dilimage
```

We go through each pixel in the original image, and search the octagonal kernel of each “white” pixel and turn them into white, which forms a dilation image.

Q2：Erosion on a binary image

```
def erosion(image, kernel):
    lena_erosion = np.zeros(image.shape, np.uint8)
    rows, columns = lena_erosion.shape
    for i in range(rows):
        for j in range(columns):
            flag = True
            for position in kernel:
                p, q = position
                if 0 <= (i + p) < 512 and 0 <= (j + q) < 512
                    and image[i + p, j + q] == 0:
                    flag = False
                    break
            if flag:
```

```
            lena_erosion[i][j] = 255
        else:
            lena_erosion[i][j] = 0
    return lena_erosion
```

We go through each pixel in the original image, and search the octagonal kernel of each pixel and turn them into white pixel when all the pixels in the kernel are all white, which forms a erosion image.

Q3：Opening on a binary image

```
def opening(image, kernel):
    return dilation(erosion(image, kernel), kernel)
```

$$A \circ B = (A \ominus B) \oplus B,$$

According to the formula mentioned above we can get the opening image by calling the erosion function first and call the dilation function.

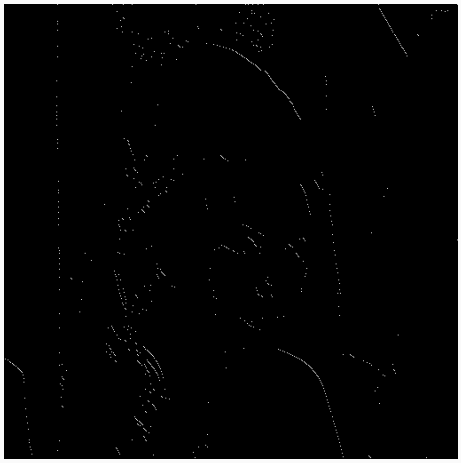
Q4：Closing on a binary image

```
def closing(image, kernel):
    return erosion(dilation(image, kernel), kernel)
```

$$A \bullet B = (A \oplus B) \ominus B,$$

According to the formula mentioned above we can get the closing image by calling the dilation function first and call the erosion function.

Q5：Hit-and-miss transform on a binary image



```
def hit_and_miss(image, jker, kker):
    image_complement = 255 - image
    eroj = erosion(image, jker)
    erok = erosion(image_complement, kker)
    ham_image = np.zeros(image.shape, np.uint8)
    for i in range(len(image)):
        for j in range(len(image[i])):
            ham_image[i][j] = eroj[i][j] & erok[i][j]
    return ham_image
```

$$A \odot B = (A \ominus C) \cap (A^c \ominus D),$$

According to the formula mentioned above, and with the L shaped kernel(jker and kker), we get the hit_and_miss image by finding the intersection of the image with erosion(binary, Lshaped kernel) and image with erosion(binary_complement, disjoint kernel).