Computer Vision Homework 4

B03902042 宋子維

Description

In this homework, we are asked to do morphological dilation, erosion, opening, closing on a gray-scale image. *Figure 1-1* is the gray-scale image in this homework, which is lena.bmp. For dilation, erosion, opening and closing, which need only one kernel, we use the 3-5-5-5-3 kernel with value 0 as shown in *Figure 1-2*.

Figure 1-1: Binary image.

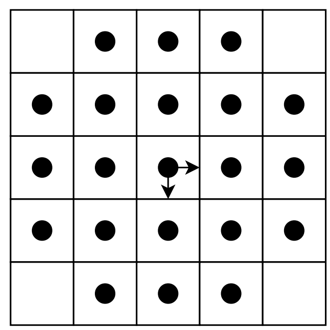

Figure 1-2: 3-5-5-5-3 kernel.

Programming

I use python to implement the algorithms. There is one python program, namely, **ImgProcess.py**, where I use **pillow** to process basic image I/O. In the program, there are some basic functions:

1. `PIL.Image.open(img)`: load the image `img` and return a pillow **Image** object.

2. `pix = Image.load()`: return the **PixelAccess** object of **Image** object to `pix`, which offers us to use `pix[x, y]` to access the pixel value at position (x, y).

3. `PIL.Image.new(mode, size)`: create a new image with given mode and size, and return a piilow **Image** object.

4. `Image.size`: pair (width, height) of **Image** object.

5. `sumTuple((a, b), (c, d))`: return the tuple (a+c, b+d).

The usage of **ImgProcess.py** is

```
python3 ImgProcess.py OPTION IMG_IN IMG_OUT
```

where **OPTION** can be **dilation**, **erosion**, **opening** and **closing**. The program will do **OPTION** transform on **IMG_IN** and generate the image named **IMG_OUT**.

Algorithm

Dilation: $A \oplus B$

Figure 2: Dilation on gray-scale image.

In the following program, we can simply regard **pix** as A, **kernel** as B and **ret** as the result, that is $A \oplus B$. Since the 3-5-5-5-3 kernel we use is with value 0 in each cell, dilation can simply be seen as local maximal within the specific window. **To do dilation, traverse each pixel (x, y) and iterate the pixels in the translation of kernel with respect to (x, y) to find the local maximal.** Then assign that value to the corresponding position of the result image.

```
def dilation(pix, size, kernel):
    width, height = size
    ret = {(x, y): BLACK for x in range(width) for y in range(height)}

    for x in range(width):
        for y in range(height):
            for _ in kernel:
                X, Y = sumTuple((x, y), _)
                if 0 <= X < width and 0 <= Y < height:
                    ret[x, y] = max(ret[x, y], pix[X, Y])

    return ret
```

Erosion: $A \ominus B$

Figure 3: Erosion on gray-scale image.

In the following program, we can simply regard **pix** as A, **kernel** as B and **ret** as the result, that is $A \ominus B$. Since the 3-5-5-5-3 kernel we use is with value 0 in each cell, erosion can simply be seen as local minimal within the specific window. **To do dilation, traverse each pixel (x, y) and iterate the pixels in the translation of kernel with respect to (x, y) to find the local minimal.** Then assign that value to the corresponding position of the result image.

Note: When the pixel in the translation of kernel is out of boundary, I do not regard it as the background (BLACK), and I directly pass it. Thus, there is not any black line along the boundary in the output image.

```
def erosion(pix, size, kernel):
    width, height = size
    ret = {(x, y): WHITE for x in range(width) for y in range(height)}

    for x in range(width):
        for y in range(height):
            for _ in kernel:
                X, Y = sumTuple((x, y), _)
                if 0 <= X < width and 0 <= Y < height:
                    ret[x, y] = min(ret[x, y], pix[X, Y])

    return ret
```

Opening: $A \circ B$

Figure 4: Opening on gray-scale image.

In the previous algorithms, I have implemented functions of dilation and erosion. As for opening, we know that $A \circ B = (A \ominus B) \oplus B$. Hence, we can simply use the same algorithms above. **First do erosion on the original image, and then we can get $A \ominus B$. Finally, do dilation on $A \ominus B$, and then we get $(A \ominus B) \oplus B$, that is, $A \circ B$.**

```
def opening(pix, size, kernel):
    return dilation(erosion(pix, size, kernel), size, kernel)
```

Closing: $A \bullet B$

Figure 4: Closing on gray-scale image.

Similarly, we know that $A \bullet B = (A \oplus B) \ominus B$. Hence, we can simply use the same algorithms above. **First do dilation on the original image, and then we can get $A \oplus B$. Finally, do erosion on $A \oplus B$, and then we get $(A \oplus B) \ominus B$, that is, $A \bullet B$.**

Note: When the pixel in the translation of kernel is out of boundary, I do not regard it as the background (BLACK), and I directly pass it. Thus, there is not any black line along the boundary in the output image.

```
def closing(pix, size, kernel):
    return erosion(dilation(pix, size, kernel), size, kernel)
```