Computer Vision Homework 4

B03902042 宋子維

Description

In this homework, we are asked to do binary morphological dilation, erosion, opening, closing, and hit-and-miss transform on a binary image. *Figure 1-1* is the binary image in this homework, which is lena.bmp with binarization at threshold 128 . For dilation, erosion, opening and closing, which need only one kernel, we use the 3-5-5-5-3 kernel as shown in *Figure 1-2*. For hit and miss transformation, it needs to kernels, namely, J and K, as shown in *Figure 1-3* and *Figure 1-4*. For each transformation, I process the white pixels.

Figure 1-1: Binary image.

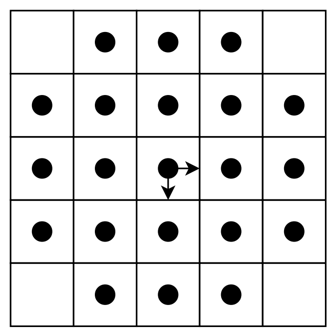

Figure 1-2: 3-5-5-5-3 kernel.

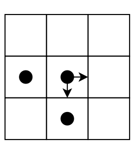

Figure 1-3: kernel J.

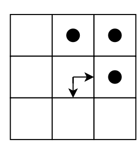

Figure 1-4: kernel K.

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**, **closing** and **hitmiss**. 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 binary 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 I process the white pixels, the first thing is to initialize each pixel with black color. **To do dilation, traverse each pixel (x, y), and dilate it with respect to 3-5-5-5-3 kernel if it is white, that is, let pixel $(x+c, y+d)$ be white for all (c, d) in kernel.**

```python
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):
            if pix[x, y] == WHITE:
                for _ in kernel:
                    keyx, keyy = sumTuple((x, y), _)
                    if keyx < 0 or keyx >= width or keyy < 0 or keyy >= height:
                        continue
                    else:
                        ret[keyx, keyy] = WHITE

    return ret
```

Erosion: $A \ominus B$

Figure 3: Erosion on binary 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 I process the white pixels, the first thing is to initialize each pixel with black color. **To do erosion, traverse each pixel (x, y), and check if all the points of (x, y) with respect to kernel, that is, $(x + c, y + d)$ for all (c, d) in kernel, are white. If they are, then pixel (x, y) is white. Otherwise, black.**

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

    total = len(kernel)
    for x in range(width):
        for y in range(height):
            sum = 0
            for _ in kernel:
                keyx, keyy = sumTuple((x, y), _)
                if keyx < 0 or keyx >= width or keyy < 0 or keyy >= height:
                    break
                elif pix[keyx, keyy] == BLACK:
                    break
                elif pix[keyx, keyy] == WHITE:
                    sum += 1
            if sum == total:
                ret[x, y] = WHITE

    return ret
```

Opening: $A \circ B$

Figure 4: Opening on binary 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):
    ret = erosion(pix, size, kernel)
    return dilation(ret, size, kernel)
```

Closing: $A \bullet B$

Figure 4: Closing on binary 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, $A \bullet B$.**

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

Hit-and-Miss: $A \otimes (J, K)$

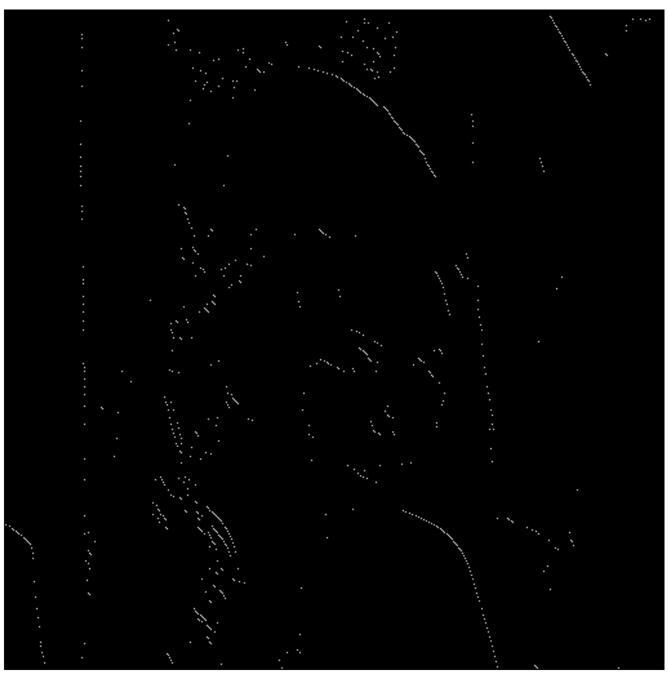

Figure 4: Hit-and-Miss on binary image.

First of all, it needs two kernels, J and K, for hit-and-miss transformation. We know that $A \otimes (J, K) = (A \ominus J) \bigcap (A^c \ominus K)$. In the following program, we can simply regard **pix** as A, **pixComplement** as A^c, **kernelJ** as J, **kernelK** as K and **ret** as the result, that is $A \otimes B$. **Since we need the complement of A, namely, A^c, the program first iterate all pixels (x, y) and assign the complement of `pix[x, y]` to `pixComplement[x, y]` (WHITE to BLACK, BLACK to WHITE). Then all materials, A, A^c, J and K, have been prepared. Similarly, we can simply use the same algorithms above, that is, `erosion()`. First, do erosion on A with respect to J, and then we can get $A \ominus J$. Second, do erosion on A^c with respect to K, and then we can get $A^c \ominus K$. Finally, intersect them. To do this, iterate all positions (x, y), and check whether the pixels at (x, y) of image $A \ominus J$ and $A^c \ominus K$ are both white. If they are, assign white color to $A \otimes (J, K)$. Otherwise, black.**

```python
def hitAndmiss(pix, size, kernelJ, kernelK):
    width, height = size
    pixComplement = {}
    for x in range(width):
        for y in range(height):
            pixComplement[x, y] = BLACK if pix[x, y] == WHITE else WHITE

    temp1 = erosion(pix, size, kernelJ)
    temp2 = erosion(pixComplement, size, kernelK)

    ret = {}
    for x in range(width):
        for y in range(height):
            if temp1[x, y] == WHITE and temp2[x, y] == WHITE:
                ret[x, y] = WHITE
            else:
                ret[x, y] = BLACK
    return ret
```