HW 4 Mathematical Morphology

Mathematical Morphology - Binary Morphology

Source code

Please refer to the file “hw4.py” within the same folder as this report document.
There are two main functions for this homework:

1. dilation(src, kernel, threshold)

```python
"""
    This function does the dilation operation.
    @param src should be an image
    @param kernel should be an instance of Kernel class
    @param threshold should be an integer (default 128 if not given)
    @return a resulted image
"""
def dilation(src, kernel, threshold=128):
    newImage = Image.new(src.mode, src.size)
    newImagePixels = newImage.load()
    for i in range(src.size[0]):
        for j in range(src.size[1]):
            old = src.getpixel((i, j))
            if old < threshold: # we're not interested in this pixel
                continue
            for direction in kernel.get_directions():
                new_i = i + direction[0]
                new_j = j + direction[1]
                if new_i >= 0 and new_i < src.size[0] and new_j >= 0 and new_j < src.size[1]:
                    # in range! set it to 255!
                    newImagePixels[new_i, new_j] = 255
    return newImage
```

2. erosion(src, kernel, threshold)

```python
"""
    This function does the erosion operation.
    @param src should be an image
    @param kernel should be an instance of Kernel class
    @param threshold should be an integer (default 128 if not given)
    @return a resulted image
"""
def erosion(src, kernel, threshold=128):
    newImage = Image.new(src.mode, src.size)
    newImagePixels = newImage.load()
    for i in range(src.size[0]):
        for j in range(src.size[1]):
            old = src.getpixel((i, j))
            set = True
            for direction in kernel.get_directions():
                new_i = i + direction[0]
                new_j = j + direction[1]
                if new_i >= 0 and new_i < src.size[0] and new_j >= 0 and new_j < src.size[1] and src.getpixel((new_i, new_j)) > threshold:
                    continue
                else:
                    set = False
                    break
            if set:
                newImagePixels[i, j] = 255
            else:
                newImagePixels[i, j] = 0
    return newImage
```

Kernel Representation

A class is defined for kernel representation:

```python
class Kernel:
    def __init__(self, init_list, origin):
        self.pattern = init_list
        self.origin = origin

    """
        @return a list of directions, i.e., list of 2-tuples
    """
    def get_directions(self):
        tmp_list = []
        for i in range(len(self.pattern)):
            for j in range(len(self.pattern[0])):
                if self.pattern[i][j] == 1:
                    direction = (j - self.origin[0], i - self.origin[1])
                    tmp_list.append(direction)
        return tmp_list
```

get_directions(self)

it returns list of directions, i.e., “vectors”, for us to traverse the kernel.

The 2 kernel patterns used in this homework

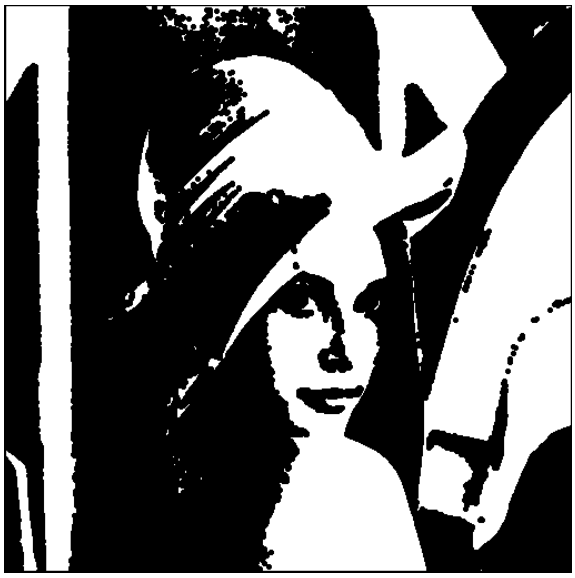
```python
octo_kernel_pattern = [
    [0, 1, 1, 1, 0],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [0, 1, 1, 1, 0]
]

L_shape_kernel_pattern = [
    [1, 1],
    [0, 1]
]
```

← We use **`octo_kernel_pattern`** with origin = (2, 2) and **`L_shape_kernel_pattern`** with both origin = (1, 0) and (0, 1).

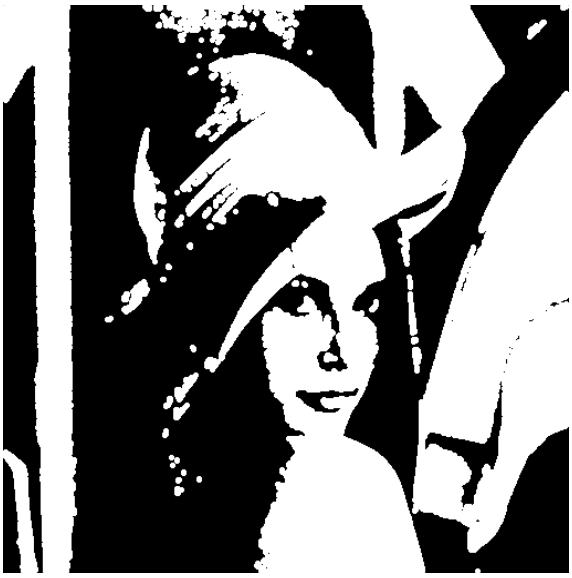
Result

(the resulted images are saved properly within the same folder as well)

↑ Dilation.bmp



↑ Erosion.bmp



↑ Opening.bmp

↑ Closing.bmp

← Hit-and-Miss.bmp