Computer Vision HW5 Report
蕭恩慈 / B07902095

The task is to write a program which does gray-scale morphology on a gray-scale image of dilation, erosion, opening, and closing.To complete the task, I use Python as my program language using numpy and PIL library.
First, define each morphological transformation function. Then in the driver program, set the kernel table to 3, 5, 5, 5, 3. Unlike the last homework which uses binary image, we will use the original image (*lena.bmp*) with center kernel [2, 2].

In dilation function, first make the original image into grayscale format, then scan each row and column of the grayscale image. For dilation, we want to find and record the local **maximum** pixel value. Set the initial value of local maximum 0, then scan each row and column in kernel, operate only on the "1" value, then calculate the position and set condition to avoid the out of range value. After getting the original pixel value, compare it to the set maximum value, then update the local maximum value. Lastly, apply the final local maximum value to the original image rows and columns.

Code snippet:

```
def dilation(ori, kernel, ctr_kernel):
        dil = Image.new("L", ori.size)
        for r in range(ori.size[0]):
        for c in range(ori.size[1]):
        px_local_max = 0
        for x in range(kernel.shape[0]):
        for y in range(kernel.shape[1]):
                if(kernel[x, y] == 1):
                target_x = r + (x - ctr_kernel[0])
                target_y = c + (y - ctr_kernel[1])
                if((0 <= target_x < ori.size[0]) and (0 <= target_y < ori.size[1])):
                px_ori = ori.getpixel((target_x, target_y))
                px_local_max = max(px_ori, px_local_max)

        dil.putpixel((r, c), px_local_max)

        return dil
```

Result:

(gray-dilated-lena.bmp)

In erosion function, first make the original image into grayscale format, then scan each row and column of the grayscale image. For erosion, we want to find the record the local **minimum** pixel value. Set the initial value of local minimum to 0, then scan each row and column in kernel, operate only on the "1" value, then calculate the position and set condition to avoid the out of range value. After getting the original pixel value, compare it to the set minimum value, then update the local minimum value. Lastly, apply the final local minimum value to the original image rows and columns.

Code snippet:

```
def erosion(ori, kernel, ctr_kernel):
        eros = Image.new("L", ori.size)
        #print(ori)
        for r in range(ori.size[0]):
        for c in range(ori.size[1]):
        px_local_min = 255
        for x in range(kernel.shape[0]):
        for y in range(kernel.shape[1]):
                if(kernel[x, y] == 1):
                target_x = r + (x - ctr_kernel[0])
                target_y = c + (y - ctr_kernel[1])
                if((0 <= target_x < ori.size[0]) and (0 <= target_y < ori.size[1])):
                px_ori = ori.getpixel((target_x, target_y))
                px_local_min = min(px_ori, px_local_min)

        eros.putpixel((r, c), px_local_min)

        return eros
```

Result:

(gray-erosion-lena.bmp)

In opening and closing functions basically involve both dilation and erosion functions. The difference is, for the opening function, it checks for the union of all translations of y that fit entirely within x, while closing is the complement of the opening function. So in the opening function, run the erosion function within the dilation function, then for closing, run the dilation function within the erosion function.

Code snippet:

```
def opening(ori, kernel, ctr_kernel):
        return dilation(erosion(ori, kernel, ctr_kernel), kernel, ctr_kernel)
```

```
def closing(ori, kernel, ctr_kernel):
        return erosion(dilation(ori, kernel, ctr_kernel), kernel, ctr_kernel)
```

Result:

(gray-opening-lena.bmp)

(gray-closing-lena.bmp)

Driver code snippet:

```
def main():
        ctr_kernel = (2, 2)
        kernel = np.array([
        [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]])

        ori = Image.open("lena.bmp")

        dil = dilation(ori, kernel, ctr_kernel)
        eros = erosion(ori, kernel, ctr_kernel)
        op = opening(ori, kernel, ctr_kernel)
        cl = closing(ori, kernel, ctr_kernel)

        dil.save("gray-dilated-lena.bmp")
        eros.save("gray-erosion-lena.bmp")
        op.save("gray-opening-lena.bmp")
        cl.save("gray-closing-lena.bmp")
```