R13922136 周洛君

Computer Vision Homework 4: Report

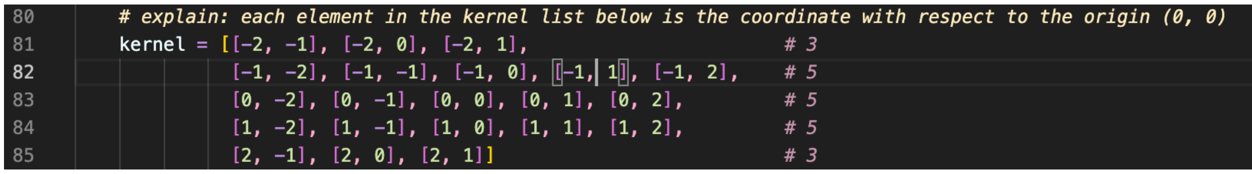
Main Code Structure

The structure of my code consists of two parts:

The first is to implement the operations for problem (a) dilation, (b) erosion, and (e) hit-and-miss transform. For the other operations (which are (c) opening and (d) closing), they're implemented simply by calling dilation and erosion, just in the opposite order, therefore they're only written in the main function.

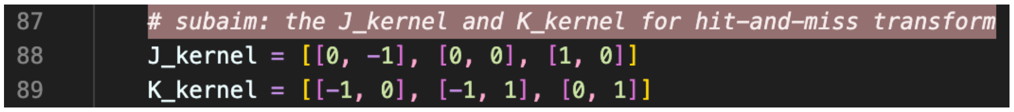
The second part is the main function, which consists of three subparts:

1. Import the grayscale lena image as usual, then transform it to a binarized image as we've done before.
2. Define the kernels, which has 2 subparts for different use:
    i. the octogonal 3-5-5-5-3 kernel (which is used in all of the problem parts except (e))



```
    # explain: each element in the kernel list below is the coordinate with respect to the origin (0, 0)
    kernel = [[-2, -1], [-2, 0], [-2, 1],                           # 3
              [-1, -2], [-1, -1], [-1, 0], [-1, 1], [-1, 2],        # 5
              [0, -2], [0, -1], [0, 0], [0, 1], [0, 2],             # 5
              [1, -2], [1, -1], [1, 0], [1, 1], [1, 2],             # 5
              [2, -1], [2, 0], [2, 1]]                              # 3
```

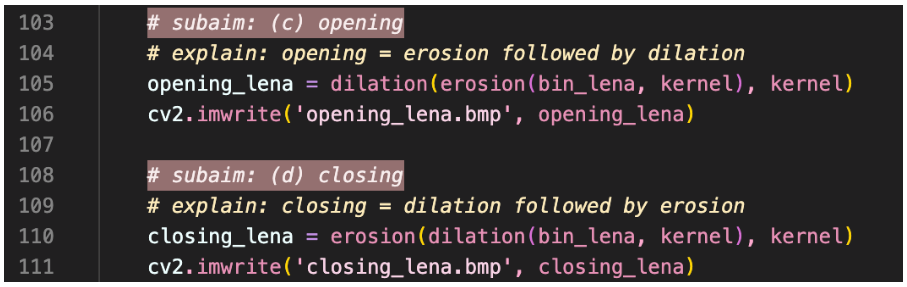
    ii. the J_kernel and K_kernel representing the "L" shaped kernel for (e) hit-and-miss transform



```
    # subaim: the J_kernel and K_kernel for hit-and-miss transform
    J_kernel = [[0, -1], [0, 0], [1, 0]]
    K_kernel = [[-1, 0], [-1, 1], [0, 1]]
```

3. The last part is to directly call the functions we implemented before, or call two of the functions to get the desired result. After that, all of the resulting image are the variables named in the form "operation name" + "lena". We then save these results by using "cv2.imwrite()".

    We deal with problem (c) and (d) as follows:

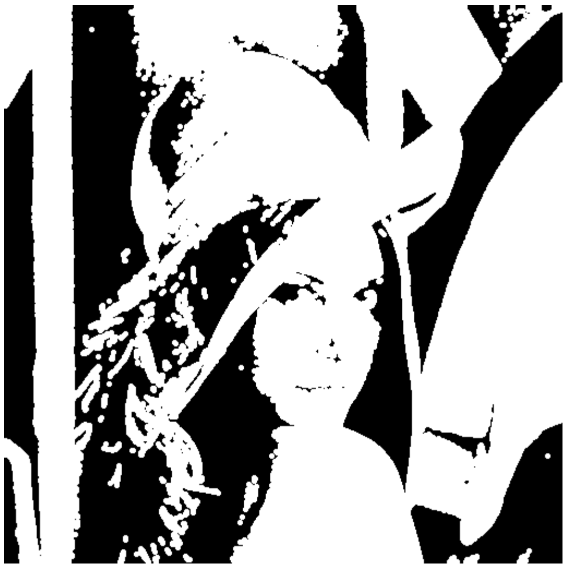


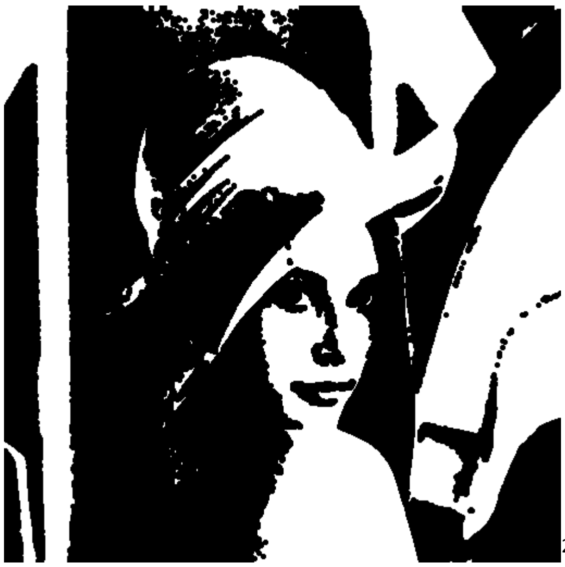
```
    # subaim: (c) opening
    # explain: opening = erosion followed by dilation
    opening_lena = dilation(erosion(bin_lena, kernel), kernel)
    cv2.imwrite('opening_lena.bmp', opening_lena)

    # subaim: (d) closing
    # explain: closing = dilation followed by erosion
    closing_lena = erosion(dilation(bin_lena, kernel), kernel)
    cv2.imwrite('closing_lena.bmp', closing_lena)
```

In the opening part, we first call “erosion()” and use the returned output as the input of “dilation()”. For the closing part, we do similar things conversely.
To save the result of problem (a) and (b) is quite simple, so just check for the content in the source code. The last problem (e) is similar, just note that we change the kernel to two kernels “J_kernel“ and “K_kernel” while calling.

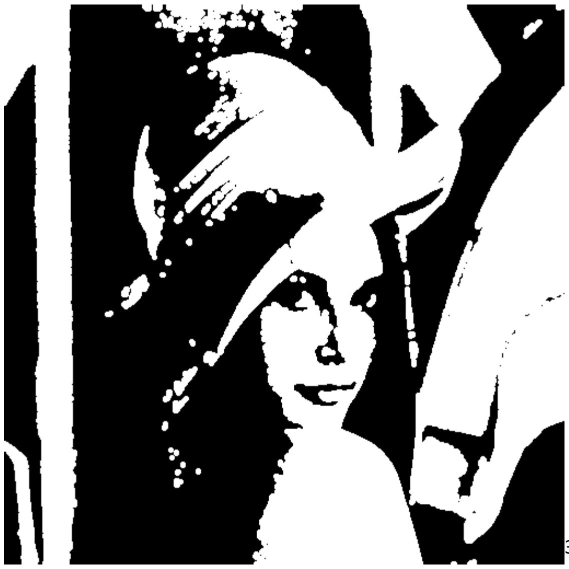
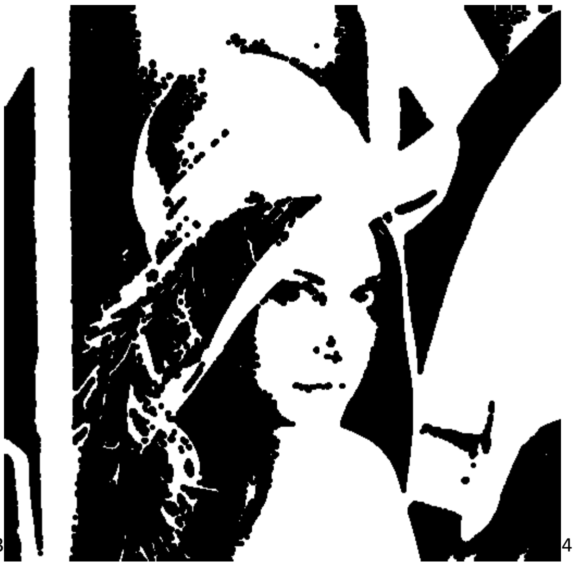
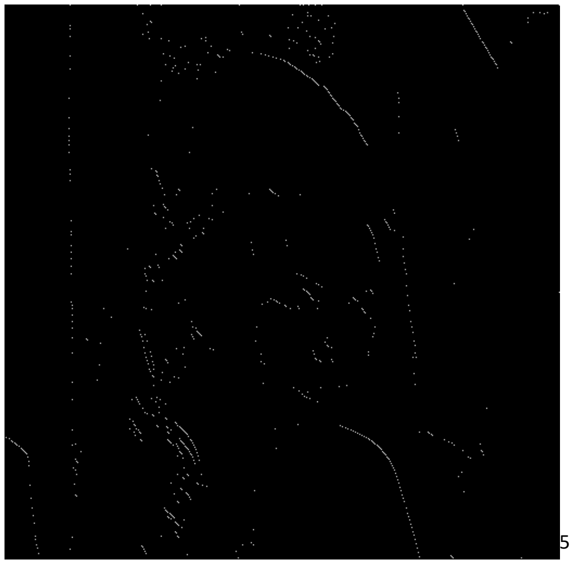
Resulting Images

The images below are the results of operation (a) to (e), and the footnote after each image refers to its name corresponding to the name we set in “cv2.imwrite()”:


[1]


[2]

[1] (a) dilation_lena.bmp
[2] (b) erosion_lena.bmp


[3]

[4]

[5]

Function Explanations:

Dilation:

In this part, we first initialize the result “dilation_img” to all zeros, then we loop through all the pixels. After that, as I stated in the comments, if we encounter any pixel that has value 255, we set the corresponding pixel in the resulting “dilation_img” also to be 255. The next thing is to set all the pixels with coordinate (ni, nj) to 255, where ni, ny are the coordinates that were the

[3] opening_lena.bmp
[4] closing_lena.bmp
[5] hit_and_miss_lena.bmp

original pixel (i,j) with every translation by each element (x,y) in the kernel. The only thing we need to check is to see if the translated coordinate is in the image. More details are written in the comments.

Erosion:

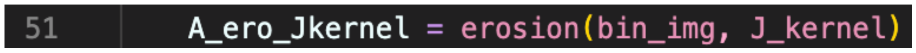
This part is the same as the previous part, except that after we loop over each element in the kernel, and check if the coordinate is in the image after translation is done, if there’s any pixel “bin_img[ni][nj]” , which is a pixel in the original binarized image, has value 0, then it means that it’s not a fit, so we set the current coordinate (i,j) in the resulting image (i.e. “erosion_img[i][j]”) to zero, and otherwise it’ a fit so set it to 255.

Hit-and-Miss transform

In this part, I use the original definition to accomplish our goal:

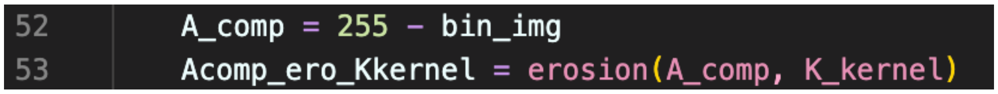
$$A \otimes (J,K) = (A \ominus J) \cap (A^c \ominus K)$$

We can see that in the two pairs of parentheses, we both do erosions, so to implement $(A \ominus J)$, I just simply use:



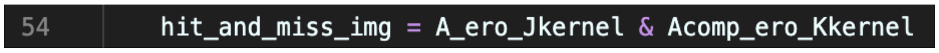
```
    A_ero_Jkernel = erosion(bin_img, J_kernel)
```

And for $(A^c \ominus K)$, we first get the complement of A, then call “erosion()” again:



```
    A_comp = 255 - bin_img
    Acomp_ero_Kkernel = erosion(A_comp, K_kernel)
```

The result would be the intersection of these two results, and this could simply be done by:



```
    hit_and_miss_img = A_ero_Jkernel & Acomp_ero_Kkernel
```

The explanation above is simplified in order to meet the page limitation, so for more details, please check for the comments, especially the comments with “explain:” in the front, these will give more insights about the meaning behind my code.