Spatial Transformations and Filtering Report

By Zhang Hewen

Southern University of Science and Technology

E-mail: 12010342@mail.sustech.edu.cn

1. Objective of the Laboratory and the Principle pf the Algorithm

The task of this week is realizing some spatial domain processes to enhance the effect of the image. Task 1-3 belong to histogram processing, task 4 belongs to spatial filtering.

Histogram describes the distribution of the number of pixels with certain intensity, and histogram processing refers to making the histogram of the image have expected effect. This laboratory focuses on histogram equalization and histogram matching. The former one means processing image in order to get a uniform histogram and the latter one means getting a image with a specified histogram. Additionally, histogram processing can be done locally or globally.

Spatial filtering is using a predefined filter and moving it step by step over the image to do a certain operation. The laboratory need us to accomplish median filtering, that is, let the median intensity in a given size of pixels substitute the original intensity.

Through the laboratory, I find that: histogram equalization is very useful for most images and it does not need prior knowledge; histogram matching can be used when there are few values of intensity and the distribution of them is dense; filtering has many types and can be used in different conditions.

Spatial transformations and filtering are practical in enhancing images, like making it clearer to see and do other research, including medical research and astronomy research.

2. Formula

1. histogram equalization

$$s_k = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j$$

For local histogram equalization, M, N and the sumation is just for the specified neighborhood.

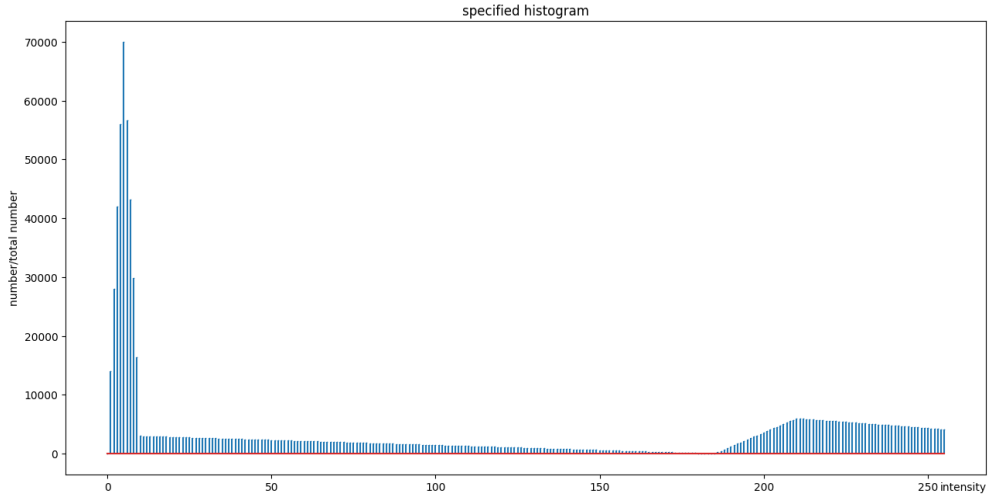
2. histogram matching

$$s_k = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j = (L-1)\sum_{i=0}^{k} p_z(z_i)$$

The specified histogram used in this lab is:

$$y = \begin{cases} 14000x,\ 0 \le y \le 5 \\ -13400x + 137000,\ 5 < y \le 10 \\ -\frac{120}{7}x + \frac{22200}{7},\ 10 < y \le 185 \\ 240x - 44400,\ 185 < y \le 210 \\ -\frac{1120}{27}x + \frac{132400}{9},\ 210 < y \le 255 \end{cases}$$

Its graph is:



> **Fig. 1.** the graph of the specified histagram for Q2

3. spatial filtering

$$g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

3. Pseudo Code

1. pseudo code for histogram equalization

```
for i in range(0, 255 + 1):
    input_hist[i] <- np.sum(input == i)
    /*the ith component of input histogram equals to the sum of pixels which have the intensity i*/

    if np.sum(img0 == i) != 0:
        arr <- np.argwhere(input == i)
        /*arr is an array consisting of the coordinates of the pixels which have the intensity i*/
```

```
        for j in range(0, len(arr)):
            output[all_coordinates_in_arr] <- 255*np.sum(input_hist)/input_size
            /*the coordinates in arr are corresponding to the output image*/
            /*the assignment follows the formula mentioned*/

input_hist <- input_hist/input_size
/*normalization*/

for j in range(0, 255 + 1):
    output_hist[j] <- np.sum(img1 == j)/output_size
    /*calculates the histogram of output*/
```

2. pseudo code for histogram matching

```
for i in range(0, 255 + 1):
    input_hist[i] <- np.sum(input == i)
    /*store the number of pixels which have intensity 0 to i*/

    equalized_input[i] <- 255*np.sum(input_hist)/input_size
    equalized_output[i] <- 255*np.sum(spec_hist[:i+1])/input_size

input_hist <- input_hist/input_size

for i in range(0,255+1):
    for k in range(i,-1,-1):
        if equalized_input[k] in equalized_output:
            arr0 <- np.argwhere(equalized_output == equalized_input[k])
            break
    /*find the nearest index in output which has the corresponding intensity to input*/

    arr1 <- np.argwhere(input == i)
    for j in range(0,len(arr1)):
        output[all_coordinates_in_arr1] <- arr0[0][0]
```

3. pseudo code for local histogram equalization

```
pad_image <- np.pad(input,(size_after_padding))
/*padding the initial image to get a larger image*/

for i in range(0 + pad_num, input.shape[0] + pad_num):
    for j in range(0 + pad_num, input.shape[1] + pad_num):
        subimage0 <- pad_image[(i-pad_num):(i+pad_num+1),(j-pad_num):(j+pad_num+1)]
        /*subimage0 is a patch of the initial image with the size requested*/

        center <- input[i-pad_num][j-pad_num]
        /*center is the maximum intensity when calculate sum of the pixels*/

```

```
        sub_sum <- 0

        for p in range(0,subimg0.shape[0]):
            for q in range(0,subimg0.shape[1]):
                if subimage0[p][q] <= center:
                    sub_sum <- sub_sum+1

        output[i-pad_num][j-pad_num] <- 255*sub_sum/filter_size
```

4. pseudo code for reducing salt-and-pepper noise

```
pad_image <- np.pad(input,(size_after_padding))

for i in range(0 + pad_num,input.shape[0] + pad_num):
    for j in range(0 + pad_num,input.shape[1] + pad_num):
        subimage0 <- input[(i-pad_num):(i+pad_num+1),(j-pad_num):
(j+pad_num+1)]
        /*choose the patch of the same size as the filter*/

        output[i-pad_num][j-pad_num] <- np.median(subimage0)
        /*find the median intensity in the filter*/
```

4. Results

1. historam equalization

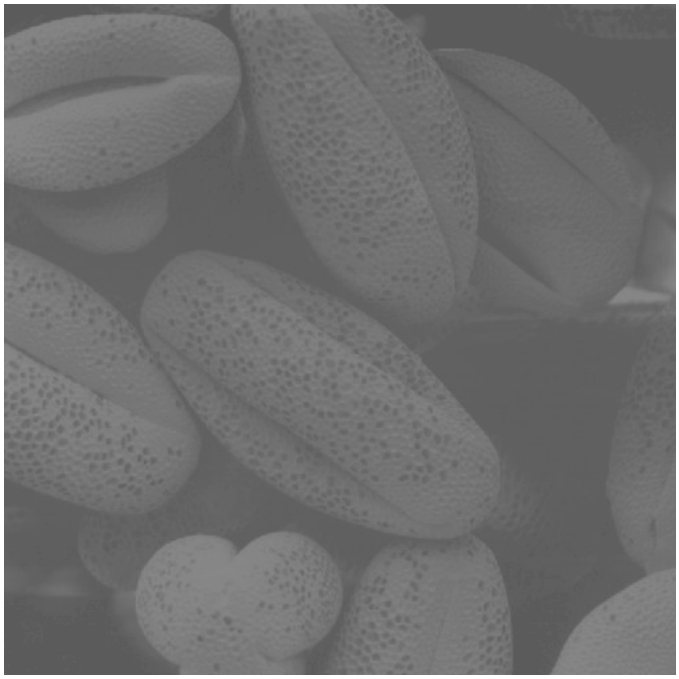


Fig. 2. the original image 1-1 with size 500*500

Fig. 3. the processed image for image 1-1 after histogram equalization

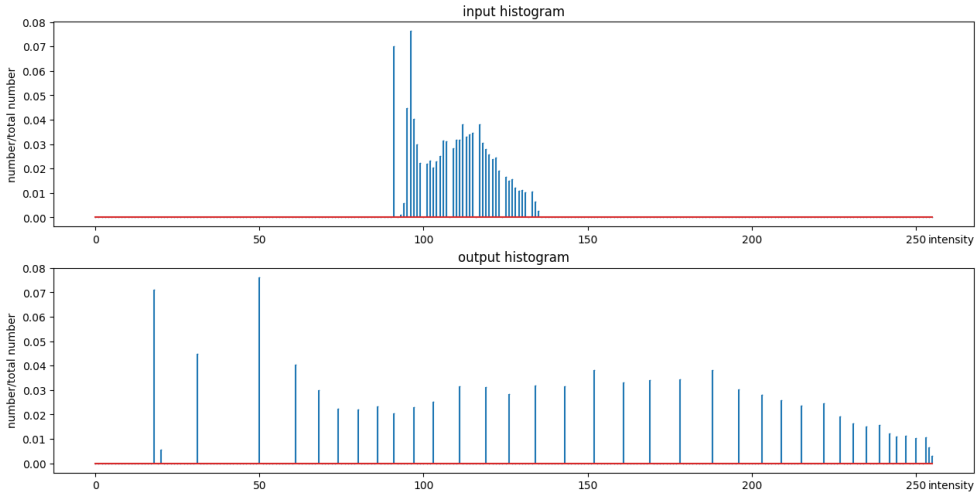


Fig. 4. the comparison of the histogram of Fig. 1. and Fig. 2.

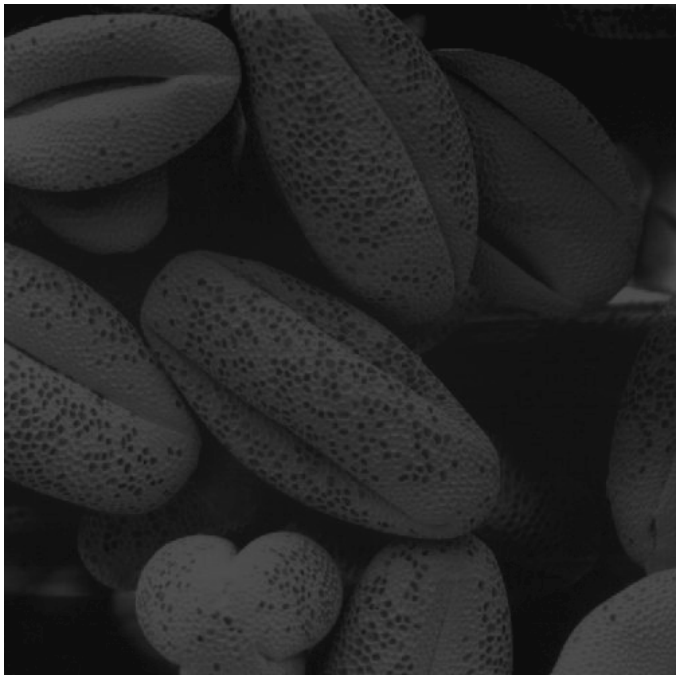


Fig. 5 the original image 1-2 with size 500*500

Fig. 6. the processed image for image 1-2 after histogram equalization

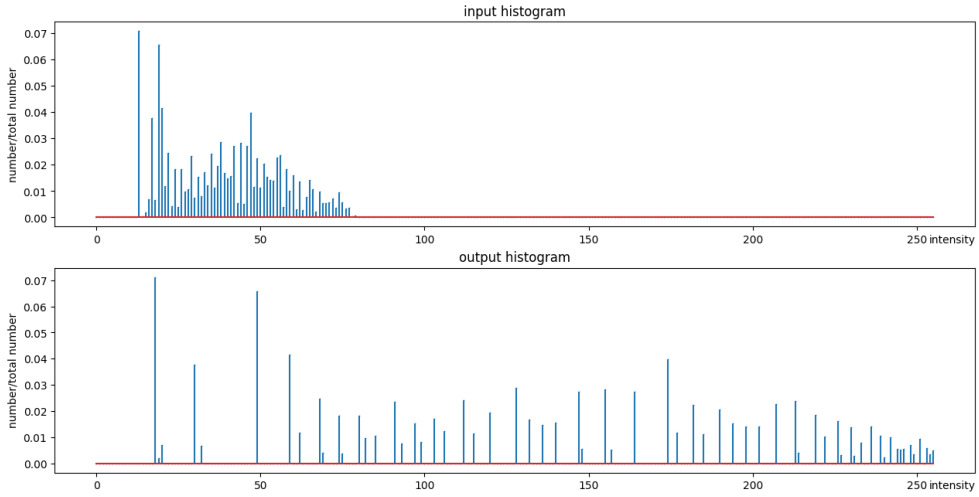


Fig. 7. the comparison of the historam of Fig. 4. and Fig. 5.

2. histogram matching

Fig. 8. the original image for Q2 of size 1000*683

Fig. 9. the processed image for Q2 after doing histogram matching

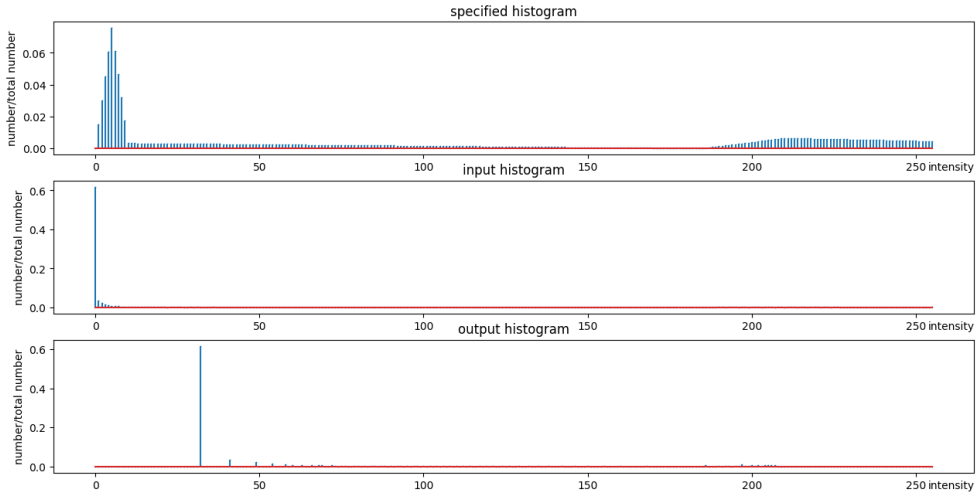


Fig. 10. the comparison of the histogram of the specified one and the input, output image

3. local histogram equalization

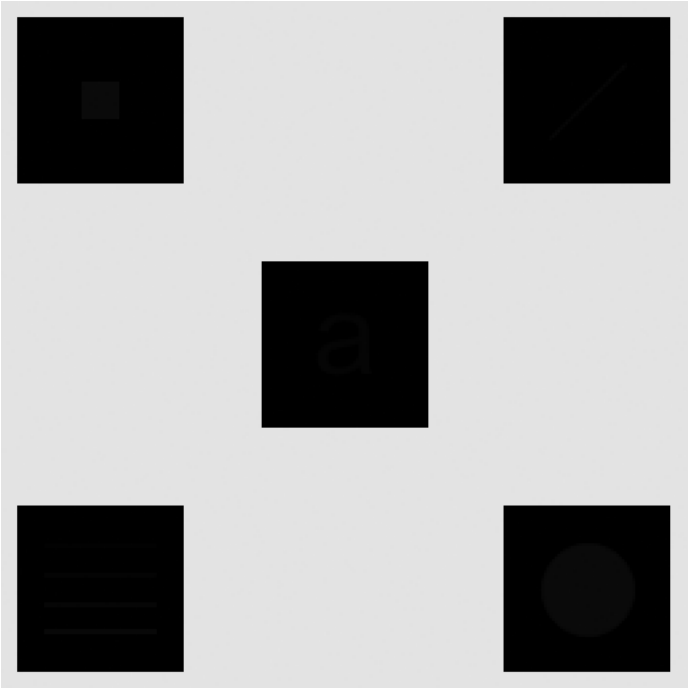


Fig. 11. the original image for Q3 of size 512*512

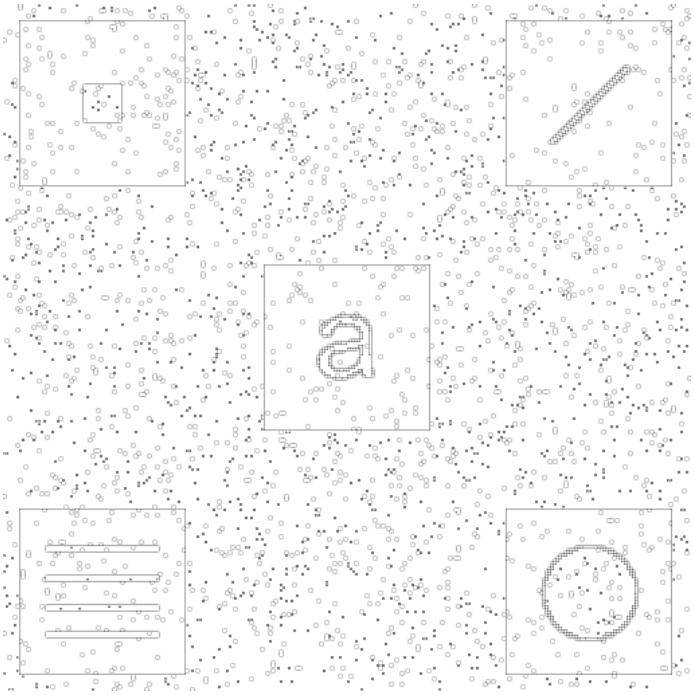


Fig. 12. the processed image for Q3 after local histogram equalization (using a neighborhood of size 3*3)

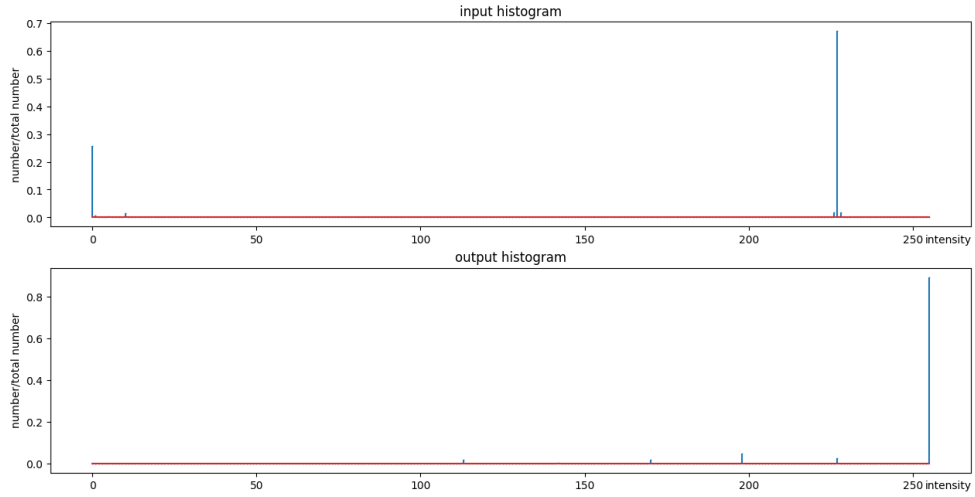


Fig. 13. the comparison of the histogram of the input and output image

4. reduce SAP

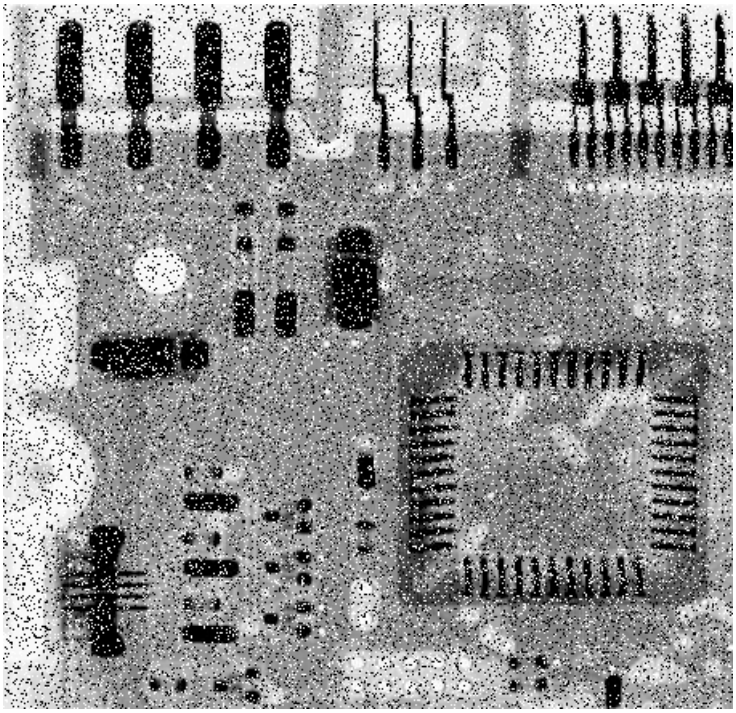
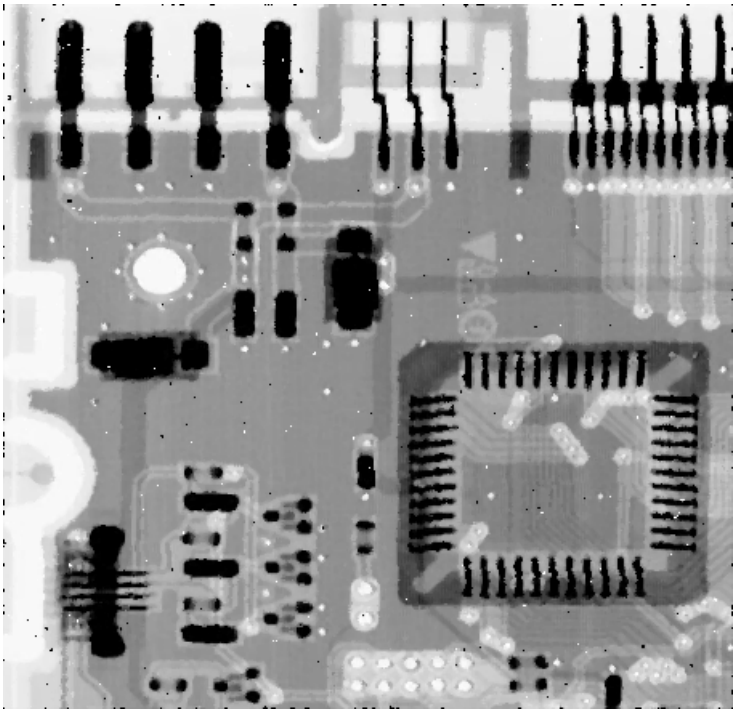


Fig. 14. the original image for Q4 of size 440*445



> **Fig. 15.** the processed image for Q4 after median filter (using a neighborhood of size 3*3)

5. Analysis

1. histogram equalization

For Q1-1 and Q1-2, the input histograms are dense, causing the image difficult to be viewed. Histogram equalization has significant effect on making images easier to distinguish. According to Fig. 4. and Fig. 7., the contrast of the image increases a lot, and the distribution of intensity becomes more uniform since being stretched.

However, there are some unnatural parts, for example, the right of the output image has strange gray color. The reason may be that the dark parts of the initial image occupy too much and the histogram can not change the amount of pixels having the same intensity.

2. histogram matching

Most pixels of the initial image has low intensity, making the image difficult to do histogram equalization to get a satisfied result. As a consequence, histogram matching is used and the target histogram is illustrated above. The result image is brighter and has more details than the original image.

There are still some parts like the boundary between white and gray which looks unreal. This because the number of pixels with low intensity is too large and the output histogram is stretched, which means that there are few high intensity values for output image. To solve the problem, I try smoothing spatial filter to enhance the image, but it made little progress.

Fig. 16. Fig. 9. after being smoothed

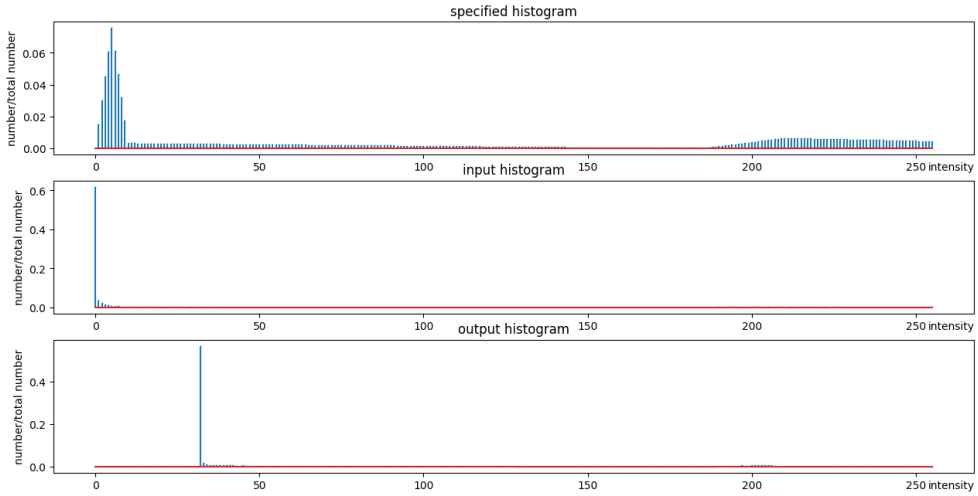


Fig. 16. the comparison of the histogram between specified one, the input and Fig.16. for Q2

3. local histogram equalization

The initial image has five black blocks and white background. It is difficult to do global histogram equalization on it directly since the intensity is discrete and not diverse. By using local histogram equalization, the hiden symbols in the image are clearly showed despite some noises.

Through changing the size of the neighborhood, it is found that the best size is 3x3. The following figures show the output image when the size is 4x4.

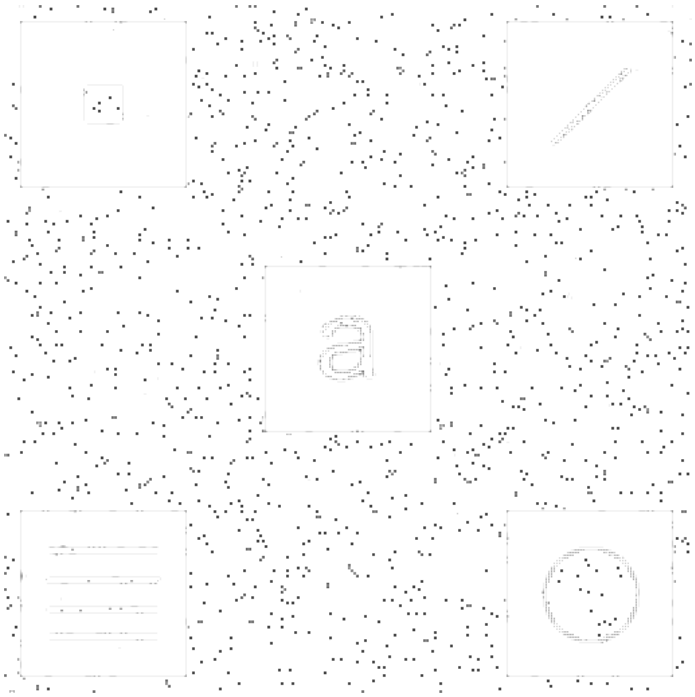


Fig. 17. the processed image for Q3 after local histogram equalization (using a neighborhood of size 4*4)

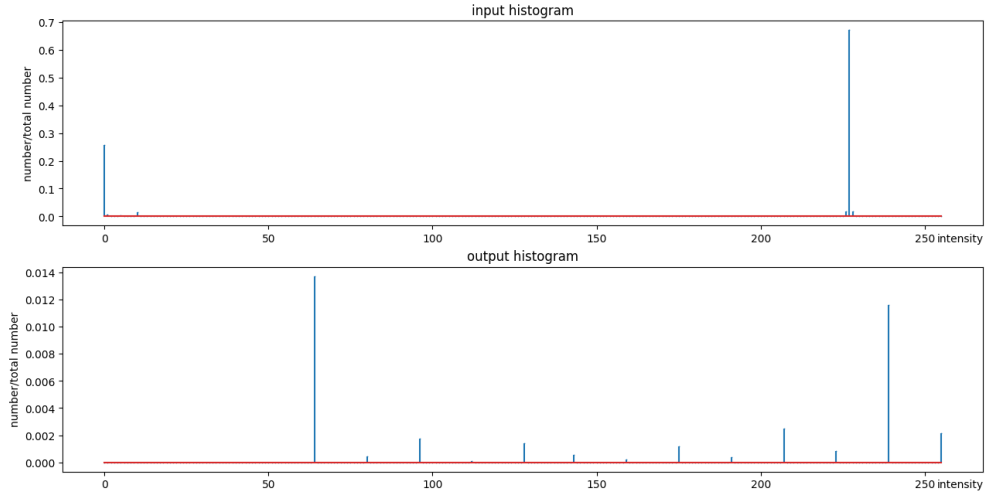


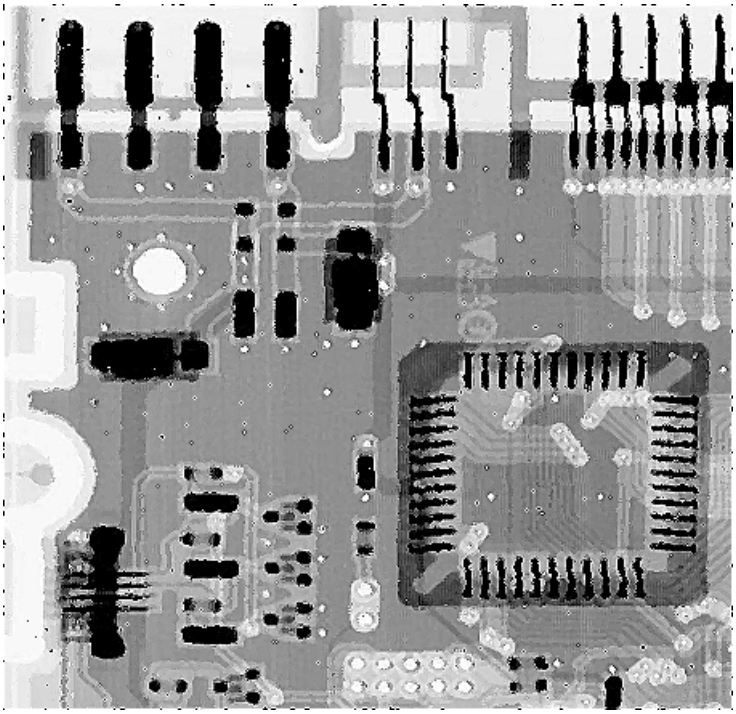
Fig. 18. the histogram for Fig. 16.

Obviously, the image is too white and the symbols are too vague. Not like Fig. 13., Fig. 17. has two high lines, which means that the noises are highlighted and the symbols are faded. This ought to the reason why the effect is not good.

It is deduced that bigger size of the neighborhood is not necessarily better.

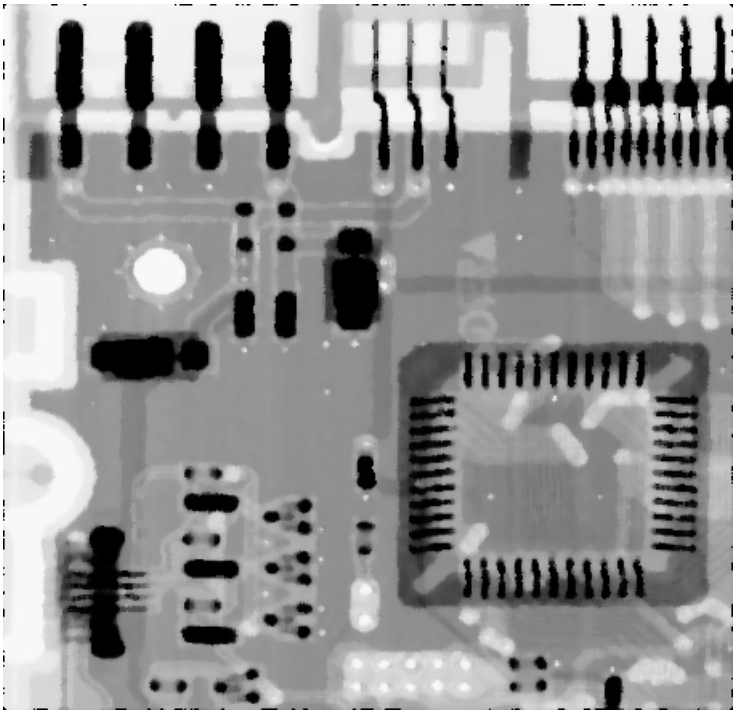
4. reduce SAP

There are a great many of salt-and-pepper noises in the original image. After trying some methods, it turns out that just utilizing median filter is enough and the size of the filter do not need to be large. A filter size of 3x3 is enough. When the size of the neighborhood is 3x3, there are still some noises, but the boundaries are clear and the details are not distorted.



> **Fig. 19.** the processed image for Q4 after median filter and sharpening spatial filter (using a neighborhood of size 3*3)

I think the result of only using a median filter of size 3*3 has vague boundaries, so I try to add a sharpening spatial filter. The result does not make great progress although being clearer.



> **Fig. 20.** the processed image for Q4 after median filter and sharpening spatial filter (using a neighborhood of size 4*4)

When processed after a median filter with a size 4*4, the image loses some details and looks vague.

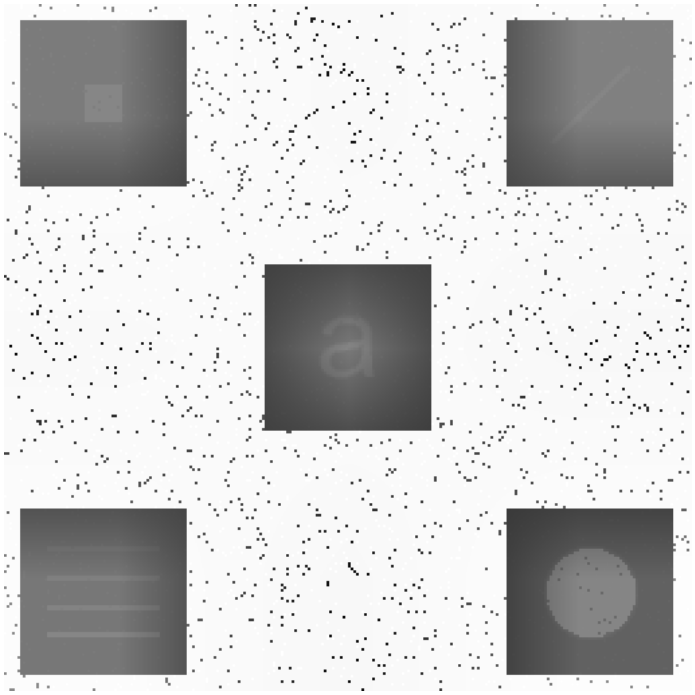
5. time cost

	my algorithm	cv2
histogram equalization	1.38s	0.04s
histogram matching	1.12s	
local histogram equalization	1.07s	0.01s
reduce SAP	2.55s	0.02s

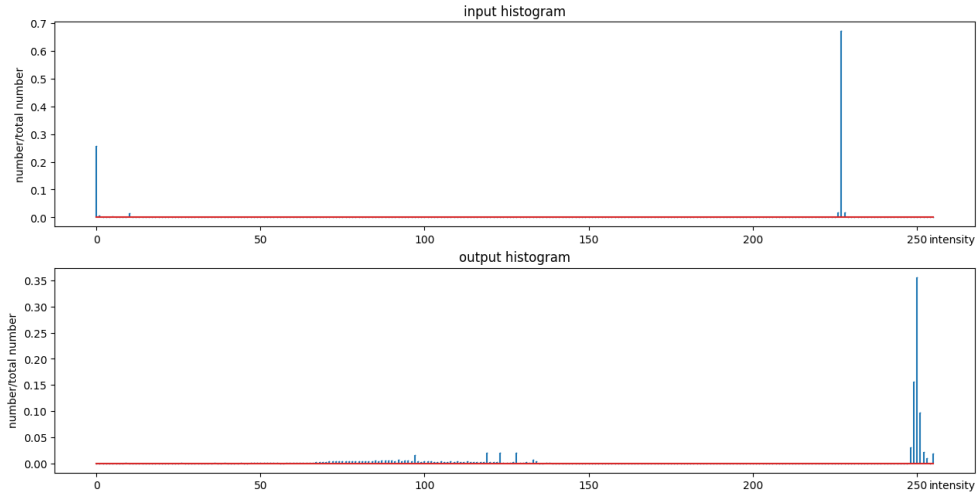
I did not find direct way of histogram matching using cv2, so the time cost of it is not written.

The algorithm of doing median filtering costs most time. This tells that shorter code may cost more time. Algorithmic complexity for histogram equalization, histogram matching, local histogram equalization and reduce SAP is $O(n), O(n), O(n^2), O(n^2)$ respectively. It causes doing median filtering to have the largest time cost.

Also, the methods in cv2 has a high efficiency, but it still has its limit. I find that the output images in Q1 by using `cv2.equalizeHist()` has a small difference with mine: my images are brighter and have lower contrast. In addition, for Q3, `cv2.createCLAHE(clipLimit= , tileGridSize= ).apply()` can not generate the expected image. Maybe invoking methods from packages still has its limit.



> **Fig. 21.** the processed image for Q3 after using the method from cv2 (using a neighborhood of size 3*3)



> **Fig. 22.** the histogram of the initial image for Q3 and Fig. 22.

6. Conclusion

The laboratory of this week aims to do basic image enhancement. I learn to write the algorithms of global and local histogram equalization, histogram matching and median filtering. From this, I have the ability to process the histogram of an image and use most kinds of filters to process images. I also know how to choose different methods to quickly improve the effect of an image.