

# Laboratory 2 Spatial Transforms and Filtering

11810506 JIA Jiyuan

2021/03/07

## Objective:

1. Implement the histogram equalization.
2. Specify a histogram, implement the specified histogram matching to the input image.
3. Implement the local histogram equalization to the input images.
4. Implement an algorithm to reduce the salt-and-pepper noise of an image.

**Keywords:** histogram equalization, histogram matching, local histogram equalization, noise reduction

## Introduction:

Spatial threshold which is different to the image processing in the transform threshold, refers to the image itself, and this kind of image processing method is based on the pixel operation in the image. The processing of spatial threshold transformation is mainly divided into gray scale transformation and spatial filtering. Grayscale transformation operates on a single pixel of the image, mainly for contrast and threshold processing purposes. Spatial filtering involves the operation of improving performance by processing a sharpened image through each pixel area of the image.

We will talk about histogram equalization, histogram matching, local histogram equalization and mediate filtering in this lab.

Histogram equalization is a method to adjust contrast using image histogram in the field of image processing. This method is often used to increase the local contrast of many images, especially when the contrast of the useful data of the image is quite close. In this way, brightness can be better distributed on the histogram. This can be used to enhance local contrast without affecting the overall contrast. Histogram equalization achieves this function by effectively extending the commonly used luminance

In image processing, histogram matching is the transformation of an image so that its histogram matches a specified histogram. The well-known histogram equalization method is a special case in which the specified histogram is uniformly distributed.

The above two methods are global, sometimes we need to enhance the details of some small local areas. So, we pick one pixel and find its neighbors, then we do the histogram equalization in this small area to find the wanted value of picked pixels.

Noise is generated because the gray value of some pixels in the image has changed, making it out of harmony with the surrounding area. Denoising removes high-frequency noise, making the gray value of the noise pixel in the image less abrupt. Median filter is a kind of nonlinear filter, which can eliminate extreme noise better, but it also brings the problem of fuzzy image.

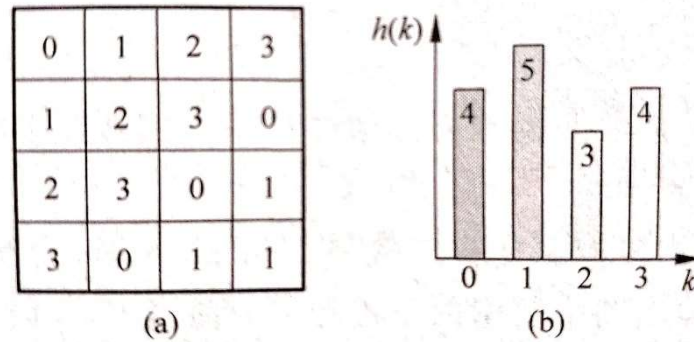
## Histogram equalization:

### Principle:

Histogram equalization is a simple and effective image enhancement technique, which changes the gray level of each pixel in the image by changing the histogram of the image. It is mainly used to enhance the contrast of the image with small dynamic range. The original image is not clear because its gray distribution may be concentrated in a narrow range. For example, the gray levels of an overexposed image are concentrated

in the high brightness range, while underexposure causes the gray levels of the image to be concentrated in the low brightness range. By using histogram equalization, the histogram of the original image can be transformed into a uniform distribution (equalization) form to increase the dynamic range of the gray value difference between pixels, to achieve the effect of enhancing the overall contrast of the image. Histogram equalization, in other words, the basic principle is the number of pixels in the image of grey value (that is, a major role on the image grey value) for broadening, and a small number of pixels of grey value (that is, does not play a major role on the picture of the grey value) to merge, thus increasing the contrast, make the image clear, reach the purpose of enhancing.

#### Question formulation:



Example of image and histogram pair

Fig. (a) is an image, and its gray histogram can be expressed as Fig. (b), where the horizontal axis represents each gray level of the image and the vertical axis represents the number of pixels of each gray level in the image. The gray histogram of the image is a one-dimensional discrete function, which can be written as:

$$h(k) = n_k \quad k = 0, 1, 2, \dots, L-1$$

The height of each column of the histogram (called bin) corresponds to  $n_k$ . The histogram provides the distribution of various gray values in the original image, or it can be said that the histogram gives the overall description of all gray values of an image. The mean and variance of the histogram are also the mean and variance of the image gray level.

The normalized histogram's relative frequency of gray level is:

$$P_r(k) = n_k/N \quad k = 0, 1, 2, \dots, L-1$$

$N$  represents the total number of pixels in image  $f(x, y)$ , and  $n_k$  is the number of pixels with gray level  $k$  in image.

$R$  and  $S$  respectively represent the normalized original image gray level and the histogram equalized image gray level

When  $r = s = 0$  the color is black,  $r = s = 1$  the color is white,  $r = s \in (0, 1)$ , the pixel gray scale changes between black and white.

For any  $r$  in the interval  $[0, 1]$ , the transformation function  $s = T(r)$  can produce a corresponding  $s$ .

$T(r)$  shall satisfy the following two conditions:

Within  $0 \leq r \leq 1$ ,  $T(r)$  is a monotonically increasing function.

Within  $0 \leq r \leq 1$ ,  $0 \leq T(r) \leq 1$

The inverse change function is  $r = T^{-1}(s)$ , which also follows the restriction above.

According to the probability theory if the random variable  $r$ 's PDF is  $p_r(r)$ , and  $s$  is the function of  $r$ , then the PDF  $p_s(s)$  can be calculated from  $p_r(r)$ .

Let  $F_s(s)$  be the CDF of  $s$ , the definition of CDF tells that

$$F_s(s) = \int_{-\infty}^s p_s(s)ds = \int_{-\infty}^r p_r(r)dr$$

If we derivative both side of  $F_s(s)$ , we can get

$$p_s(s) = \frac{dF_s(s)}{ds} = \frac{d[\int_{-\infty}^r p_r(r)dr]}{ds} = p_r(r) \frac{dr}{ds} = p_r(r) \frac{dr}{d[T(r)]}$$

For a uniform distribution over the interval  $[0, L - 1]$ , the probability density function is  $p_s(s) = \frac{1}{L-1}$ .

Normalization will make the PDF be  $p_s(s) = \frac{1}{1-0} = 1$ .

We let  $p_s(s) = 1$  and integral both sides to get

$$s = T(r) = \int_0^r p_r(r)dr$$

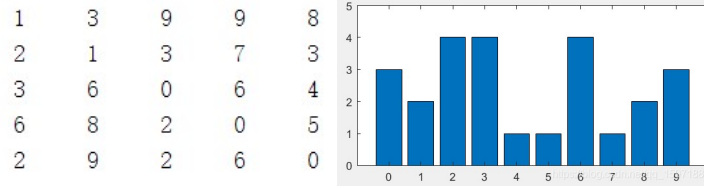
For digital images with discrete gray levels, frequency is used to replace probability, then the function is transformed to

$$s_k = T(r_k) = \sum_{i=0}^k p_r(r_i) = \sum_{i=0}^k \frac{n_i}{N}$$

$$0 \leq r_k \leq 1, k = 0, 1, 2, \dots, L - 1$$

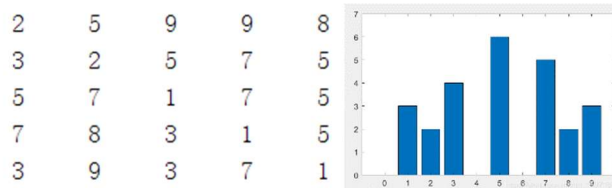
$r_k = \frac{k}{L-1}$  represent the normalized gray level,  $k$  is the gray level of original image. Similarly,  $s_k$  is also a normalized gray level, we can multiply it with  $L - 1$  to get the unnormalized gray level.

We can consider the original image shown as this



Example of original image and its histogram

1. Count the total number of original images.
2. Calculate the gray level histogram  $n_k$  of original image. It is obvious that  $n(0) = 3$ , so we can arrange  $n(1), n(2), n(3), \dots, n(9)$  to be a list as  $n(k) = [3 \ 2 \ 4 \ 4 \ 1 \ 1 \ 4 \ 1 \ 2 \ 3]$ .
3. Get the frequency of gray level distribution  $p_r(k) = \frac{n_k}{N} = \left[ \frac{3}{25} \ \frac{2}{25} \ \frac{4}{25} \ \frac{4}{25} \ \frac{1}{25} \ \frac{1}{25} \ \frac{4}{25} \ \frac{1}{25} \ \frac{2}{25} \ \frac{3}{25} \right]$ ,  $k = 0, 1, 2, \dots, 9$ .
4. Get the cumulative frequency of gray level distribution  $s_k = \sum_{i=0}^k \frac{n_i}{N} = \left[ \frac{3}{25} \ \frac{5}{25} \ \frac{9}{25} \ \frac{13}{25} \ \frac{14}{25} \ \frac{15}{25} \ \frac{19}{25} \ \frac{20}{25} \ \frac{22}{25} \ \frac{25}{25} \right]$ ,  $k = 0, 1, 2, \dots, 9$ .
5. The normalized  $s_k$  should be multiplied by  $L - 1$  and rounded to make the gray level of the equalized image consistent with the original image before normalization.
6. According to the above mapping relationship and referring to the pixels in the original image, the histogram equalized image can be written as



The example image after processing and its histogram

After equalization, there are three gaps, by which the original adjacent gray values are expanded, thus

enlarging the contrast. However, merging also reduces the contrast of some adjacent pixels like row 4 column 5, the above one is changed from 5 to 4.

#### Experiment:

```
1. from typing import io
2. import numpy as np
3. from skimage import io
4. import matplotlib.pyplot as plt
5.
6.
7. def hist_equ_11810506(image_input):
8.     """ Histogram equalization of a grayscale image. """
9.     input_img = io.imread(image_input)
10.    r, c = input_img.shape
11.    output_img = np.zeros([r, c], dtype=np.uint8)
12.    input_histogram = []
13.    output_histogram = []
14.
15.    # histogram of input image
16.    # pdf
17.    for i in range(256):
18.        input_histogram.append(np.sum(input_img == i) / (r * c))
19.
20.    # get cumulative distribution function
21.    cdf = []
22.    sum = 0
23.    for i in range(len(input_histogram)):
24.        sum = sum + input_histogram[i]
25.        cdf.append(sum)
26.
27.    # cdf = 255 * cdf / cdf[-1]
28.
29.    for i in range(r):
30.        for j in range(c):
31.            output_img[i, j] = ((256 - 1)) * cdf[input_img[i, j]]
32.
33.    for i in range(256):
34.        output_histogram.append(np.sum(output_img == i) / (r * c))
35.
36.    io.imsave(image_input.strip(".tif") + "_11810506.tif", output_img)
37.
38.    n = np.arange(256)
39.    plt.plot(n, input_histogram)
40.    plt.savefig(image_input.strip(".tif") + "_input_hist_11810506.tif")
```

```

41. plt.close()
42. plt.plot(n, output_histogram)
43. plt.savefig(image_input.strip(".tif") + "_output_hist_11810506.tif")
44. plt.close()
45.
46.
47. return (
48.     image_input + "_11810506.tif", image_input + "output_hist_11810506.t
    if",
49.     image_input + "input_hist_11810506.tif")
50.
51.
52. hist_equ_11810506("Q3_1_1.tif")
53. hist_equ_11810506("Q3_1_2.tif")

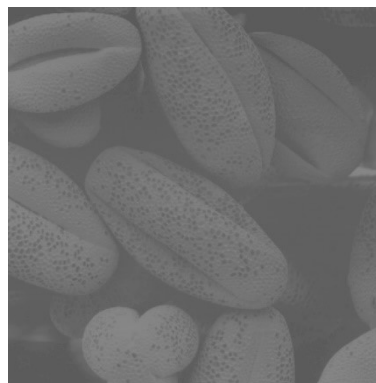
```

when calculate CDF, `numpy.cumsum(PDF)` will decrease the time about 0.5s.

## Results and analysis:

### Results:

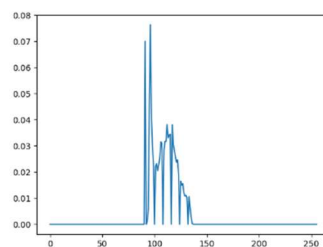
Input image 1



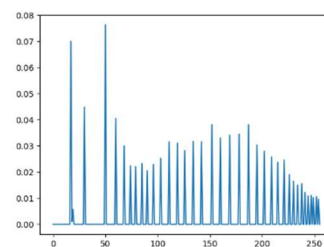
Output image 1



Input image 1's histogram

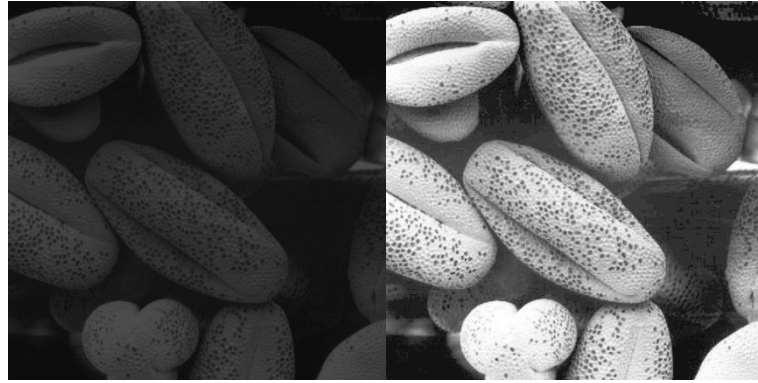


Output image 1's histogram



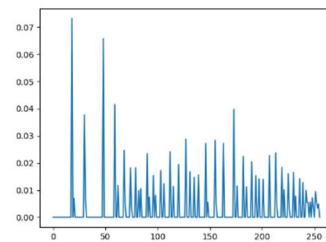
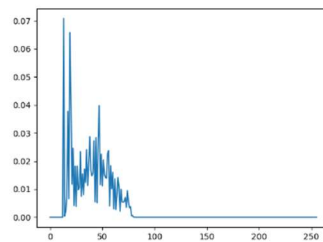
Input image 2

Output image 2



**Input image 2's histogram**

**Output image 2's histogram**



#### **Analysis:**

**Effect:** This approach generally increases the global contrast of many images. It works well for images with backgrounds that are too bright or too dark, allowing for better distribution of intensity across the histogram. This allows areas with local low contrast to obtain high contrast. The equalized image can better show the structural details of pollen.

**Advantages:** This method is useful in images where the background and foreground are both bright and dark. In particular, this approach allows for a better view of bone structure in X-ray images and better detail in over-exposed or under-exposed photos. A key advantage of this method is that it is a simple technique and an invertible operator. So, theoretically, if the histogram equalization function is known, the original histogram can be restored.

**Disadvantages:** The disadvantage of this method is that it is indiscriminate. It may increase the contrast of the background noise while reducing the available signal, detail will be lost and increase the unnatural excess.

#### **Histogram matching:**

##### **Principle:**

For some specific application histogram equalization is not always a best method. In most cases, we may want to use some special histogram to process the image to obtain the hypothetical effect. Histogram matching is an enhancement method that transforms the histogram of an image into the histogram of a specified shape. The key is using a gray image function, make the original gray histogram into the desired histogram.

##### **Question formulation:**

We let two continuous gray level be  $r$  and  $z$ .  $p_r(r)$  and  $p_z(z)$  will be their PDF function. Variable  $r$  and  $p_r(r)$  are attributes of input image,  $z$  and  $p_z(z)$  belong to the image we wanted.

We assume two random variables  $s$  and  $G(z)$ :

$$s = T(r) = (L - 1) \int_0^r p_r(r) dw$$

$$G(z) = (L - 1) \int_0^z p_z(t) dt = s$$

We find  $G(z) = T(r)$  and  $z = G^{-1}[T(r)] = G^{-1}(s)$ .

So, we can conclude the procedure to do the histogram matching:

1. Calculate the  $p_r(r)$  of input image and find the CDF to get  $s_k$ .
2. Use the  $p_z(z)$  of wanted image to get  $G(z)$  and store the integer part of value in a list.
3. For each  $s_k$ ,  $k = 0, 1, 2, \dots, L - 1$ , match the corresponding  $G(z)$  in list. If the mapping is not unique, smallest one will be the choice. Then we can get  $z$ , use  $z$  we can reconstruct a new image.

### Experiment:

```

1. from typing import io
2. from skimage import io
3. import cv2
4. import numpy as np
5. import matplotlib.pyplot as plt
6. # load original image
7. img=cv2.imread("Q3_2.tif")
8. img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
9. # load goal image
10. goal_img=cv2.imread("Q3_2_goal.tif")
11. goal_img=cv2.cvtColor(goal_img,cv2.COLOR_BGR2GRAY)
12.
13. input_histogram=[]
14. input_CDF=[]
15. input_CDF_scaled=[]
16. goal_histogram=[]
17. goal_CDF=[]
18. goal_CDF_scaled=[]
19.
20. # equalization of goal image to obtain wanted histogram
21. r, c = goal_img.shape
22. for i in range(256):
23.     goal_histogram.append(np.sum(goal_img == i) / (r * c))
24. sum = 0
25. for i in range(len(goal_histogram)):
26.     sum = sum + goal_histogram[i]
27.     goal_CDF.append(sum)
28. for i in range(256):
29.     goal_CDF_scaled.append(round(255 * goal_CDF[i]))
30.
31. # do equalizaition to origianl image
32. r, c = img.shape

```

```

33. for i in range(256):
34.     input_histogram.append(np.sum(img == i) / (r * c))
35. sum = 0
36. # CDF
37. for i in range(len(input_histogram)):
38.     sum = sum + input_histogram[i]
39.     input_CDF.append(sum)
40. # integer part
41. for i in range(256):
42.     input_CDF_scaled.append(round(255 * input_CDF[i]))
43.
44.
45. g = []
46. for i in range(256):
47.     s = input_CDF_scaled[i]
48.     flag = True
49.     for j in range(256):
50.         if goal_CDF_scaled[j] == s:
51.             g.append(j)
52.             flag = False
53.             break
54.     if flag == True:
55.         minp = 255
56.         jmin = 0
57.         for j in range(256):
58.             b = abs(goal_CDF_scaled[j] - s)
59.             if b < minp:
60.                 minp = b
61.                 jmin = j
62.         g.append(jmin)
63.
64. print(len(g))
65. for i in range(r):
66.     for j in range(c):
67.         img[i, j] = g[img[i, j]]
68.
69.
70. io.imsave("Q3_2"+"_11810506.tif", img)
71. n = np.arange(256)
72. plt.plot(n, input_histogram)
73. plt.savefig("Q3_2" + "_hist_11810506.tif")
74. plt.close()
75. plt.plot(n, goal_histogram)
76. plt.savefig("Q3_2" + "_goal_hist_11810506.tif")

```



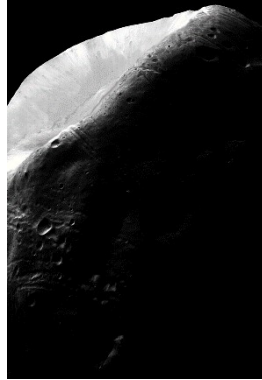
```
77. plt.close()
```

when calculate CDF, `numpy.cumsum(PDF)` will decrease the time about 0.5s.

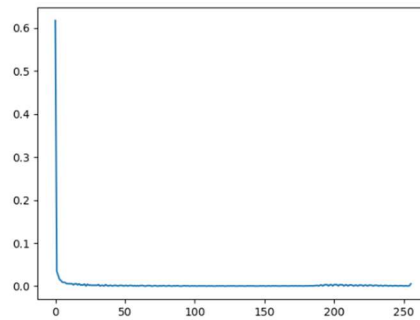
### Results and analysis:

#### Results:

Input image



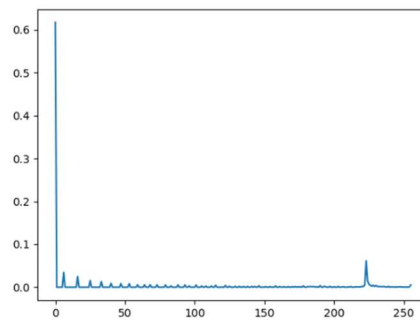
Input image's histogram



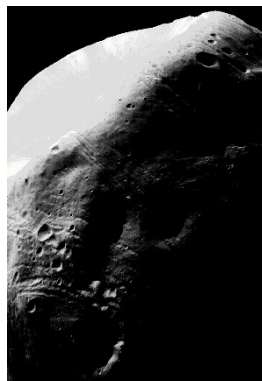
Goal image



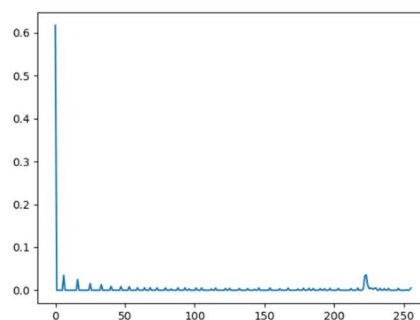
Goal image's histogram



Output image



Output image's histogram



#### Analysis:

The output image's histogram is like the goal image's histogram.

Sometimes simply histogram equalization can not give a satisfied output image, we need a special histogram distribution. By looking at the resulting image, we can clearly see that we have a specific histogram that represents more detail in the image. And using different histograms, you can get different output graphs.

The output graph is shown using histogram equalization.

### Local histogram equalization:

#### Principle:

The previous discussions are all histogram processing for the whole image, and histogram processing is also applicable to local parts. Histogram processing techniques can be used for local enhancement. The process is to define a neighborhood and move the center of the neighborhood from one pixel to another. At each position, the histogram of the midpoint in the domain is calculated, and the histogram is either equalized or defined transformation function is obtained, which is finally used to map the gray level of the center pixel in the neighborhood. The center of the field is then moved to an adjacent pixel and the process is repeated.

#### Question formulation:

The procedure to do the local histogram equalization:

1. Find the histogram in the first neighborhood.
2. The pixel of the center point of the neighborhood is updated according to histogram equalization.
3. Move the center point to the next neighborhood. For example, if the center point is (3,3) the first number is row, and the second value is column), move down one pixel first, and the center point becomes (4,3). Assuming Size=7, the neighborhood obtained at this time is different from the previous neighborhood by only one row of pixels, that is (0,0), (0,1), ..., (0,6) and (7,0), (7,1), ..., (7,6) may be different. At this time, compare whether the corresponding elements in line 0 and line 7 are the same to update the histogram. If the histogram changes, update the pixel value of the current center point.
4. Perform the third step on all the pixels.

#### Experiment:

```
1. from typing import io
2. import numpy as np
3. from skimage import io
4. import matplotlib.pyplot as plt
5.
6. def hist_equ(part):
7.     """ Histogram equalization of a grayscale image. """
8.     r, c = part.shape
9.     input_img = part.flatten().sort()
10.    x = int((r - 1) / 2)
11.    sum = 0
12.    for i in range(int(part[x, x] + 1)):
13.        sum = sum + np.sum(input_img == i) / (r * c)
14.    # get cumulative distribution function
15.    print(input_img)
16.    return (255) * sum
17.
18. def part_img(input_img, m, n, m_size):
19.    step = int((m_size - 1) / 2)
```

```

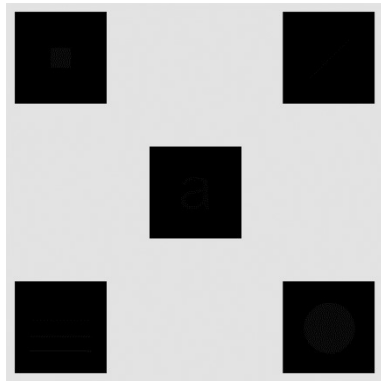
20.     part = np.zeros([m_size, m_size], dtype=np.uint8)
21.
22.     for i in range(m - step, m + step):
23.         for j in range(n - step, n + step):
24.             if i >= 0 and i < input_img.shape[0] and j >= 0 and j < input_im
                g.shape[0]:
25.                 part[i - (m - step), j - (n - step)] = input_img[i, j]
26.     return part
27.
28. def local_hist_equ_11810506(input_image, m_size):
29.     input_img = io.imread(input_image)
30.     output_img = np.zeros(input_img.shape, dtype=np.uint8)
31.     r, c = input_img.shape
32.     input_histogram = [] # Distribution of input pixels
33.     output_histogram = [] # Distribution of output pixels
34.
35.     # Count input
36.     for i in range(256):
37.         input_histogram.append(np.sum(input_img == i))
38.
39.     # local histogram equalization
40.     for i in range(0,r):
41.         for j in range(0,c):
42.             output_img[i, j] = hist_equ(part_img(input_img, i, j, m_size))
43.             #print(i,j)
44.     # Count output
45.
46.     for i in range(256):
47.         output_histogram.append(np.sum(output_img == i))
48.
49.     io.imsave("Q3_3" + "_11810506.tif", output_img)
50.
51.     n = np.arange(256)
52.     plt.plot(n, input_histogram)
53.     plt.savefig("Q3_3" + "_input_hist_11810506.tif")
54.     plt.close()
55.     plt.plot(n, output_histogram)
56.     plt.savefig("Q3_3" + "_output_hist_11810506.tif")
57.     plt.close()
58.     return
59.
60.
61. local_hist_equ_11810506("Q3_3.tif", 3)

```

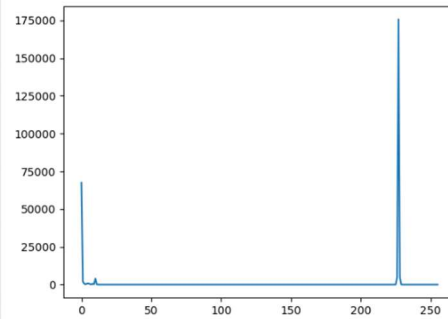
## Results and analysis:

### Results:

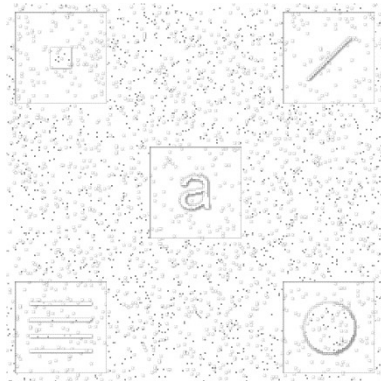
Input image



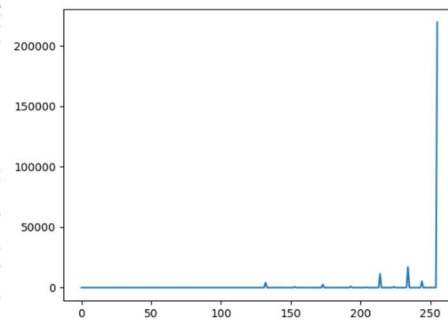
Input image's histogram



Output image



Output image's histogram



### Analysis:

By observing the resulting image, we can recognize the covered symbol under black block. Some blemishes and distortions occur, but they do not affect the detail or shape of the image. This method not only enhances the contrast of local details of the image, but also eliminates the block effect. Because the total number of sub-block equalization is equal to the total number of pixels in the input image, the efficiency of the algorithm is low.

### Improvement:

I try to involve the hist\_equ and part\_img together, and when we calculate CDF, we just calculate the number of pixels smaller than central pixel. Because we do not need to know each level's number, just total number of pixel whose level is no larger than central pixel is enough.

```
def part_img(input_img, m, n, m_size):  
    sums = 0  
    step = int((m_size - 1) / 2)  
    for i in range(m - step, m + step + 1):  
        for j in range(n - step, n + step + 1):  
            if 0 <= i < input_img.shape[0]  
            and 0 <= j < input_img.shape[1]:  
                if input_img[i, j] <=  
                input_img[m, n]:  
                    sums = sums + 1  
  
    return (255) * sums / (m_size * m_size)
```

The time needed can be decreased from 1 minute to 3s, for size equal 3.

totally cost 2.13 s

#### **Alternating method:**

Beside the method mentioned above, we can have other ways to apply. According to the overlap degree of the equalized sub-blocks, which can be divided into overlapping sub-blocks, non-overlapping sub-blocks and partial overlapping sub-blocks.

#### **Equalization algorithm for overlapping subblocks:**

The algorithm is applied above, this method not only enhances the contrast of local details of the image, but also eliminates the block effect. Because the total number of sub-block equalization is equal to the total number of pixels in the input image, the efficiency of the algorithm is low.

#### **Equalization algorithm with non-overlapping subblocks:**

The algorithm divides the input image into a series of non-overlapping sub-blocks and performs independent histogram equalization for each sub-block.

We just need to change the step in the above methods to divide the image into several area.

The advantage of the algorithm is that the contrast of local details of the image can be fully enhanced, but the disadvantage is that the histogram equalization function of each sub-block is quite different, so it is difficult to avoid the block effect in the output image.

Both the subblock non-overlapping algorithm and the subblock overlapping algorithm have great limitations. The subblock non-overlapping algorithm will lead to the inevitable block effect. Although the subblock overlapping algorithm overcomes the block effect, the subblock moves pixel by pixel, which is very inefficient.

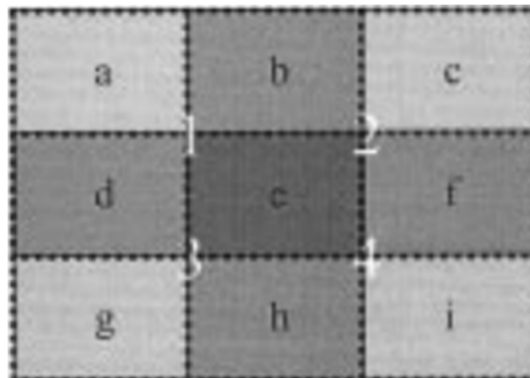
So, we have subblock partial overlapping algorithm.

#### **Subblock partial overlapping algorithm:**

##### **Principle:**

Instead of moving the sub-block pixel by pixel, the moving step is reduced to a fraction of the size of the sub-block. The gray conversion function of subblock balance is not only used to map the gray value of the center pixel of the subblock, but also used to map the gray value of all pixels of the subblock. For pixels that have been equalized for many times, the equalization result is averaged as the gray value of the pixel in the output image.

#### **Question formulation:**



This mask can be created by moving the sub block, such as a 3\*3 mask, which can be created by moving the sub block by 1/2 of its size. For a 120 by 120 subblock, the move step is 60, and you get a

3 by 3 mask. Further, if the subblock size is moved by 1/4, a 7\*7 mask can be obtained. Move 1/8 of the sub-block size to get a 15\*15 mask.....That is to say, the size of the mask can be controlled by changing the moving step size of the child block.

This is a 3\*3 mask, a-i are sub block, the block has move four time, so we have 1-4, area e has be processed 4 times, so we can take the average of POSHE.

$$s_k^e = \frac{1}{4} [T_1(r_k^e) + T_2(r_k^e) + T_3(r_k^e) + T_4(r_k^e)]$$

$$T_1(r_k^e) = \sum_{j=0}^k p_1(r_j)$$

$$p_1(r_j) = \frac{n_j^1}{\frac{4n}{9}} = \frac{n_j^a + n_j^b + n_j^d + n_j^e}{\frac{4n}{9}}$$

$$T_2(r_k^e) = \sum_{j=0}^k p_2(r_j)$$

$$p_2(r_j) = \frac{n_j^2}{\frac{4n}{9}} = \frac{n_j^a + n_j^c + n_j^e + n_j^f}{\frac{4n}{9}}$$

$$T_3(r_k^e) = \sum_{j=0}^k p_3(r_j)$$

$$p_3(r_j) = \frac{n_j^3}{\frac{4n}{9}} = \frac{n_j^d + n_j^e + n_j^g + n_j^h}{\frac{4n}{9}}$$

$$T_4(r_k^e) = \sum_{j=0}^k p_4(r_j)$$

$$p_4(r_j) = \frac{n_j^4}{\frac{4n}{9}} = \frac{n_j^e + n_j^f + n_j^h + n_j^i}{\frac{4n}{9}}$$

So we can get that:

$$\begin{aligned} p(r_k^e) &= \frac{1}{4} p_e(r_j^e) + \frac{1}{8} [p_b(r_j^e) + p_d(r_j^e) + p_f(r_j^e) + p_h(r_j^e)] \\ &\quad + \frac{1}{16} [p_a(r_j^e) + p_c(r_j^e) + p_g(r_j^e) + p_i(r_j^e)] \end{aligned}$$

We just need to change the step as part of size in the algorithm mentioned first. And filter the output image.

#### Analysis:

The advantages of subblock partial overlap algorithm are as follows:

Because the partial overlap of sub-blocks reduces the shape difference of the equalization function between adjacent sub-blocks, the block effect can basically eliminate the small amount of block effect that may appear on the boundary of sub-blocks, and it is not difficult to overcome with

the block effect elimination filter (BERF).

Because the total number of subblock equalization is much less than that of subblock overlap, the computational efficiency is greatly improved.

The enhancement ability of image details is like that of subblock overlap algorithm.

### **Salt and pepper noise and median filtering :**

#### **Principle:**

Salt-and-pepper noise is black and white bright and dark noise generated by image sensors, transmission channels, decoding processes, etc.

The so-called pepper salt, pepper is black, salt is white, pepper salt noise is a random black and white pixel on the image. Salt and pepper noise is a kind of noise caused by the intensity of signal pulse.

Salt and pepper noise is often caused by image cutting. The most used algorithm to remove pulse interference and salt and pepper noise is median filtering. Many experimental studies have found that the images captured by cameras are seriously affected by discrete pulses, salt and pepper noise and zero-mean Gaussian noise. Noise brings a lot of difficulties to image processing, which has a direct impact on image segmentation, feature extraction and image recognition. Therefore, the real-time collected images need to be filtered. Eliminating noise from the image is called image smoothing or filtering operation. There are two purposes of filtering: one is to extract the features of objects as the feature pattern of image recognition; The second is to adapt to the requirements of computer processing to eliminate the noise mixed with image digitization. There are two requirements for filtering: one is not to damage the contour and edge of the image and other important information; Two is to make the image clear, visual effect is good.

Median filter is a kind of typical nonlinear filtering technology, the basic idea is to use pixel neighborhood grey value of the median instead of the gray levels of pixels, this method in the removal of impulse noise, salt and pepper noise while preserving image edge details, median filtering is based on the theory of order statistics of a nonlinear signal processing technology can effectively restrain noise, its basic principle is to put the little value in a digital image or sequence used at various points in the field of a point at which the value of the median instead, let the surrounding pixel values close to the real value of order to remove isolated noise points and is particularly useful for speckle noise and salt and pepper noise, Because it doesn't depend on values in the neighborhood that are very different from the typical values. Median filter is similar to linear filter in processing continuous image window function, but the filtering process is no longer weighted operation.

Median filtering under certain conditions can overcome the common such as minimum mean square filter, linear filter box filter, median filter, such as details of image, and image scanning to filter out pulse interference and noise is highly effective, and is often used to protect the edge information, save the edge features to make it on the edge of the undesirable fuzzy occasion is also useful, it is very classic smooth noise processing method.

#### **Question formulation:**

Briefly introduce the procedure:

1. In a sequence of numbers  $\{1,4,6,8,9\}$ , the number 6 is the median of the sequence. Therefore, we can apply it to image processing. Again, we take a 3 by 3 matrix in the image, there are 9 pixels in it, we sort the 9 pixels, and we assign the center of the matrix to be the median of the 9 pixels.
2. 
$$g = \text{median}[f(x-1, y-1) + f(x-1, y) + f(x-1, y+1) + f(x, y-1) + f(x, y) + f(x, y+1) + f(x+1, y-1) + f(x+1, y) + f(x+1, y+1)]$$

### Experiment:

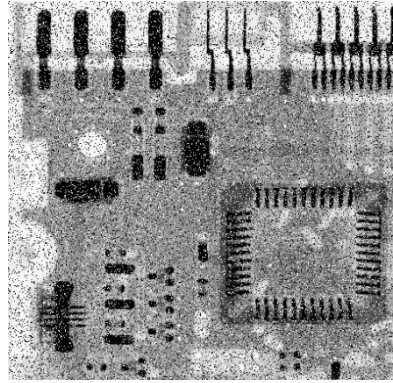
```
1. import numpy as np
2. from skimage import io
3.
4. def reduce_SAP_11810506(input_image, n_size):
5.     im=io.imread(input_image)
6.     im_copy=np.zeros(im.shape, dtype=np.uint8)
7.
8.     for i in range(0,im.shape[0]):
9.         for j in range(0,im.shape[1]):
10.            im_copy[i][j]=im[i][j]
11.     step = n_size
12.     for i in range(int(step/2),im.shape[0]-int(step/2)):
13.         for j in range(int(step/2),im.shape[1]-int(step/2)):
14.            im_copy[i][j]=m_filter(i,j,n_size,im)
15.
16.     io.imshow(im_copy)
17.     io.imsave("Q3_4_11810506.tif", im_copy)
18.     return "Q3_4_11810506.tif"
19. # filter
20. def m_filter(x,y,n_size,im):
21.     step = n_size
22.     sum_s=[]
23.     for k in range(-int(step/2),int(step/2)+1):
24.         for m in range(-int(step/2),int(step/2)+1):
25.             sum_s.append(im[x+k][y+m])
26.             sum_s.sort()
27.
28.     return sum_s[(int(step*step/2)+1)]
29.
30.
31. reduce_SAP_11810506("Q3_4.tif",3)
```

### Results and analysis:

#### Results:

##### Input image:

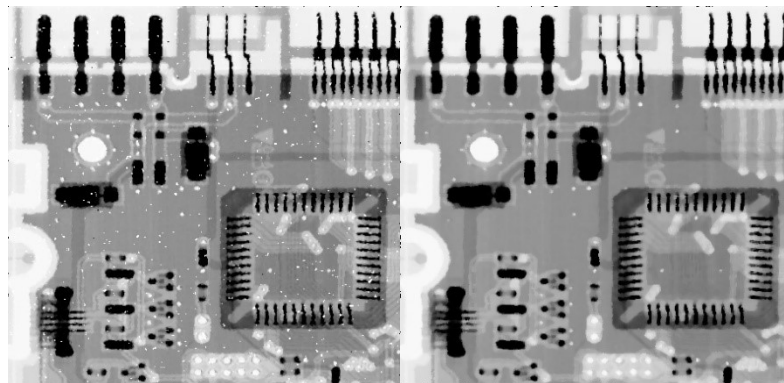




Output image:

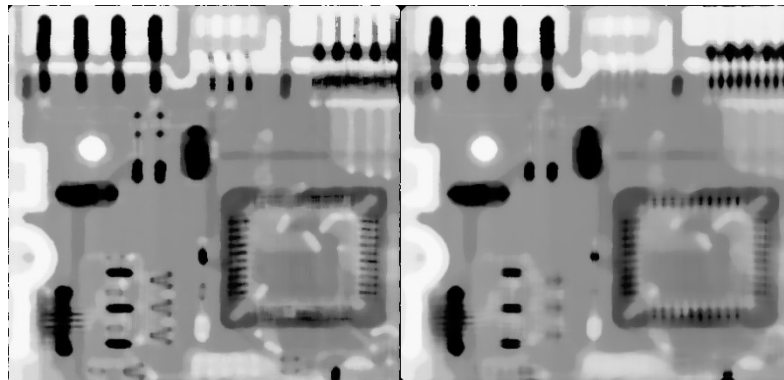
Size: 3

Size: 5



Size: 11

Size: 17




#### Analysis:

Size = 3, noise was removed a lot. But the image still has a lot of white point.

Size = 5, it seems clearer.

Is the larger the value of size the better?

Size = 11, Although the salt and pepper noise disappeared, the image also became blurred, because the size was too large, the useful information was also replaced by the median value, and the error gradually appeared

Let us go to the extreme, Size = 17, it is getting blurred. But the strangest one is  in size 17 is

better than  in size 11.

Although the larger the size (template size is size \* size) can effectively eliminate noise, but it will make the boundary blurred, so the choice of size directly affects the quality of the picture.

**Conclusion:**

In this experiment, we have implemented the histogram equalization, histogram matching, local histogram equalization and SAP reduction. Each method has its unique function, when we apply them, their advantage and shortcoming should be considered.

Histogram equalization works well for images with backgrounds that are too bright or too dark, allowing for better distribution of intensity across the histogram. This allows areas with local low contrast to obtain high contrast. The equalized image can better show the structural details of pollen. But it may increase the contrast of the background noise while reducing the available signal, detail will be lost and increase the unnatural excess.

Sometimes we need a special histogram distribution to represent more detail in the image. Histogram matching can help us to do this.

Local histogram equalization not only enhances the contrast of local details of the image, but also eliminates the block effect. Because the total number of sub-block equalization is equal to the total number of pixels in the input image, the efficiency of the algorithm is low. So, we have part overlapping methods to obtain a balance between effect and cost.

Median filtering can reduce SAP noise, but if we use over large size of block, it will make the boundary blurred, so the choice of size directly affects the quality of the picture.