Lab3 Spatial Transforms and Filtering

张旭东 12011923

1. Introduction

Intensity transform can achieve the balance of the contrast of image. The balance of the contrast of image can suppress the invalid information in the image, so that the image can be converted into the form of the processing analysis of computer or human, so as to improve the visual value and use value of the image. Spatial filtering can improve the quality of image by removing the interference of noise with high frequency and enhancing the edge of influence.

Histogram equalization, histogram matching, local histogram equalization and median filtering is the most common four methods.

Histogram equalization changes the gray value of each pixel in the image by changing the histogram of the image to enhance the contrast of images with small dynamic range. However, histogram equalization is a global processing method and is indiscriminate in processed data, which may increases the contrast of background interference information and reduce the contrast of useful pixels.

Histogram matching matches the histogram of one image to the other image so that the two images are in the same hue. When the gray level of the image is mostly concentrated in the low level, it can preserve the general shape of the image histogram.

Local histogram equalization is a local operation of histogram equalization, which maps the intensity of the pixel centered in the neighborhood. When there is a large change of gray value between bright and dark areas in the image, it can preserve the information in the places where the gray value is high

Median filtering replaces the value of a point in the digital image with the median value based on a neighborhood of the point. The larger the size of median filter, the better effect of reducing salt-and-pepper noise. However, the image is getting blurry with the size of median filter increasing because the useful information is replaced by the median.

2. Analysis and Result

2.1 histogram equalization

principle: histogram equalization is a simple and effect image enhance technique, which changes the gray value of each pixel in the image by changing the histogram of the image to enhance the contrast of images with small dynamic range. Briefly speaking, the basic principle of histogram is to widen the gray value with a large number of pixels in the image that play a major role in the image and merge gray values with a small number of pixels, which can increase contrast and make the image more clear.

Question formulation:

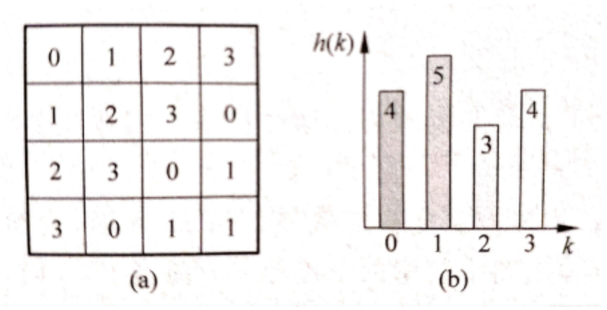


Fig.1 example of image and histogram pair

The gray histogram of an image with intensity levels $[0, L-1]$ is a one dimensional discrete function, which can be written as:

$$h(r_k) = n_k \quad k = 0, 1, 2\ldots\ldots L-1 \tag{1}$$

r_k is the k^{th} intensity value and n_k is the number of pixels in the image of size $M \times N$ with intensity r_k. The normalized histogram is denoted by $p(r_k)$, which is an estimate of the probability of occurrence of intensity level r_k in an image.

$$p(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2\ldots\ldots L-1 \tag{2}$$

For any r in the interval $[0, L-1]$, there is a transformation function $s = T(r)$ which satisfies the following conditions:

- $T(r)$ is a strictly monotonically increasing function in the interval $0 \leqslant r \leqslant L-1$
- $0 \leqslant T(r) \leqslant L-1$ for $0 \leqslant r \leqslant L-1$

The intensity levels in an image may be viewed as random variables in the interval $[0, L-1]$. Let $p_r(r)$ and $p_s(s)$ denote the probability density function of random variables r and s.

The CDF of s, $F_s(s)$ is:

$$F_s(s) = \int_0^s p_s(s)ds = \int_0^r T(r)dr \tag{3}$$

Then the formula of $p_s(s)$ is:

$$p_s(s) = \frac{dF_s(s)}{ds} = \frac{d[\int_0^r T(r)dr]}{ds} = p_r|\frac{dr}{ds}| \tag{4}$$

The reason for taking the absolute value is $p_s(s) \geqslant 0$. An transformation function of particular importance in image processing is:

$$s = T(r) = (L-1)\int_0^r p_r(r)dr \tag{5}$$

Then

$$\begin{gathered}\frac{ds}{dr} = \frac{dT(r)}{dr} = (L-1)\frac{d}{dr}[\int_0^r p_r(r)dr] = (L-1)p_r(r) \\ p_s(s) = p_r|\frac{dr}{ds}| = \frac{p_r(r)}{(L-1)p_r(r)} = \frac{1}{L-1}\end{gathered} \tag{6}$$

For discrete values:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = (L-1)\sum_{j=0}^{k}\frac{n_j}{MN} \quad k = 0, 1, 2\ldots\ldots L-1 \tag{7}$$

pseudo code:

```
1. get the histogram of input_image: $p(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2\ldots\ldots L-1$
2. get $s$: $s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$
3. map $r_k$ to $s_k$ : output_image[$i$][$j$]=$s_k$ where input_image[$i$][$j$]=$r_k$
```

The code is as below:

```
def hist_equ_12011923(input_image):
    H,W=input_image.shape
    total=H*W*1

    output_image=np.zeros([H,W],dtype=np.uint8)
    input_hist=[]
    output_hist=[]

    #input_histogram
    for i in range(256):
        input_hist.append(np.sum(input_image==i)/(total))



    sum_h=0
    for i in range(0,256):
        ind=np.where(input_image==i)
        sum_h+=len(input_image[ind])
        z_prime=round(255/total*sum_h)
        output_image[ind]=z_prime



    #output_histogram
    for i in range(256):
        output_hist.append(np.sum(output_image==i)/(total))

    return (output_image,input_hist,output_hist)
```

The function `np.sum` can return the number of pixels whose value is equal to a certain value. The function `np.where` returns the coordinates whose values are equal to a certain value and the values of corresponding coordinates in the output_image is $round(s_k)$ because the gray value of each pixel is integer.

result:

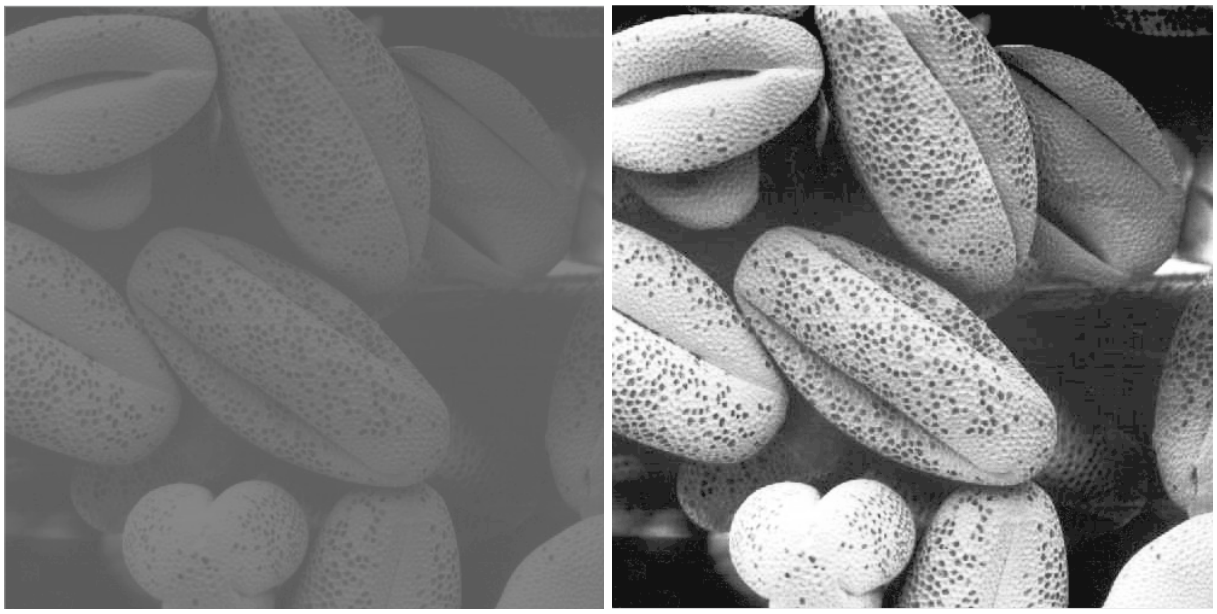


Fig.2 input_image1(left) and output_image1(right)

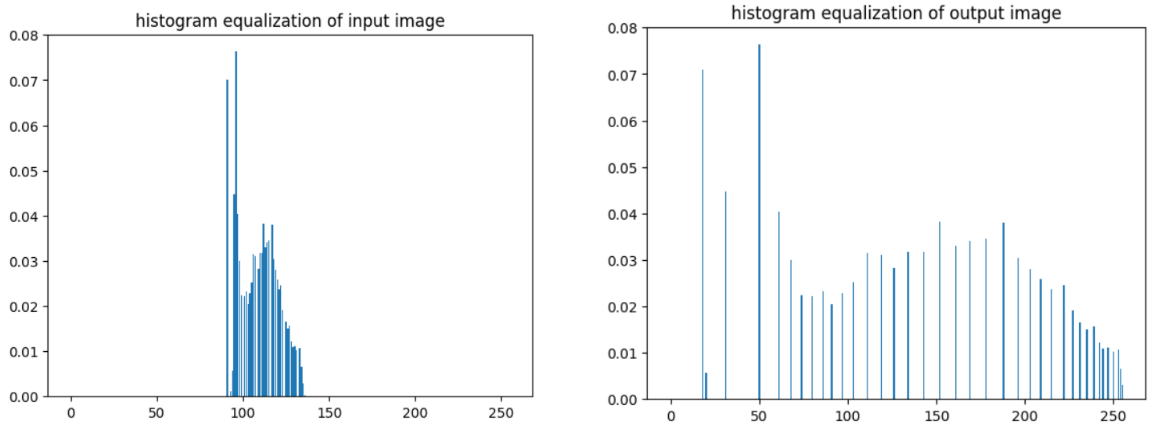


Fig.3 histogram of input_image1(left) and histogram of output_image1(right)

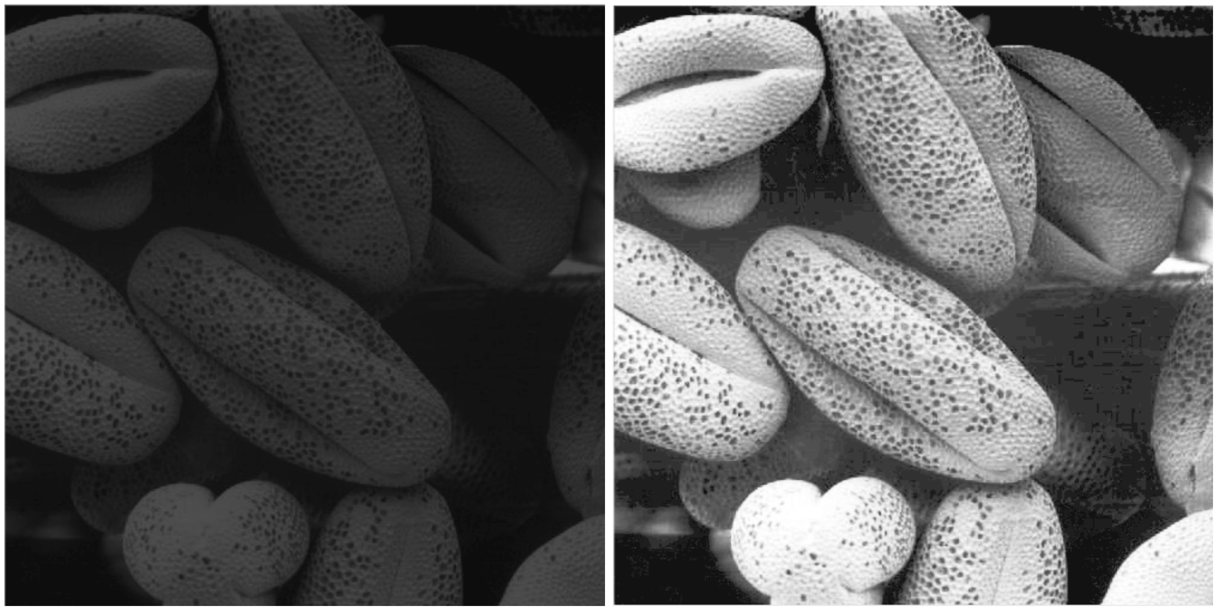
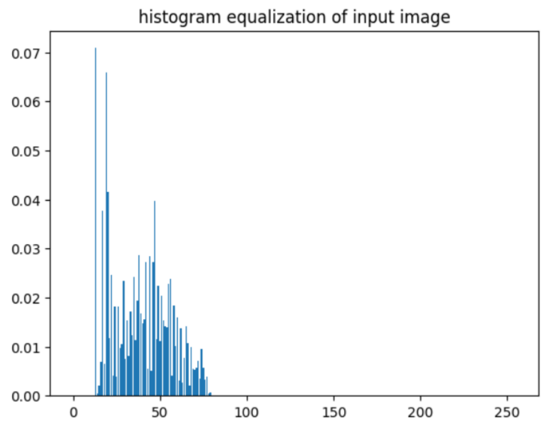


Fig.4 input_image2(left) and output_image2(right)



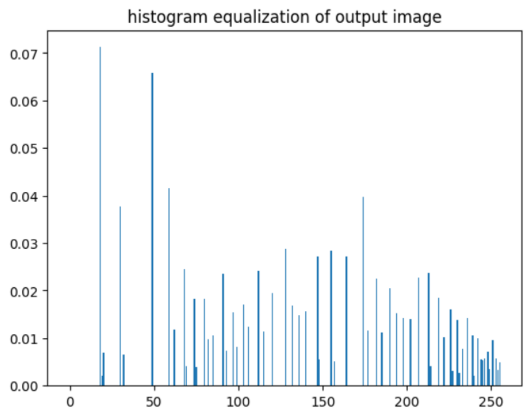


Fig.5 histogram of input_image2(left) and histogram of output_image2(right)

Analysis: From the above two output_images, it is obvious that histogram equalization increases the global contrast of the input_images. What's more, it is concluded that this method works well for images with backgrounds that are too bright or too dark, allowing for better distribution of intensity across the histogram.

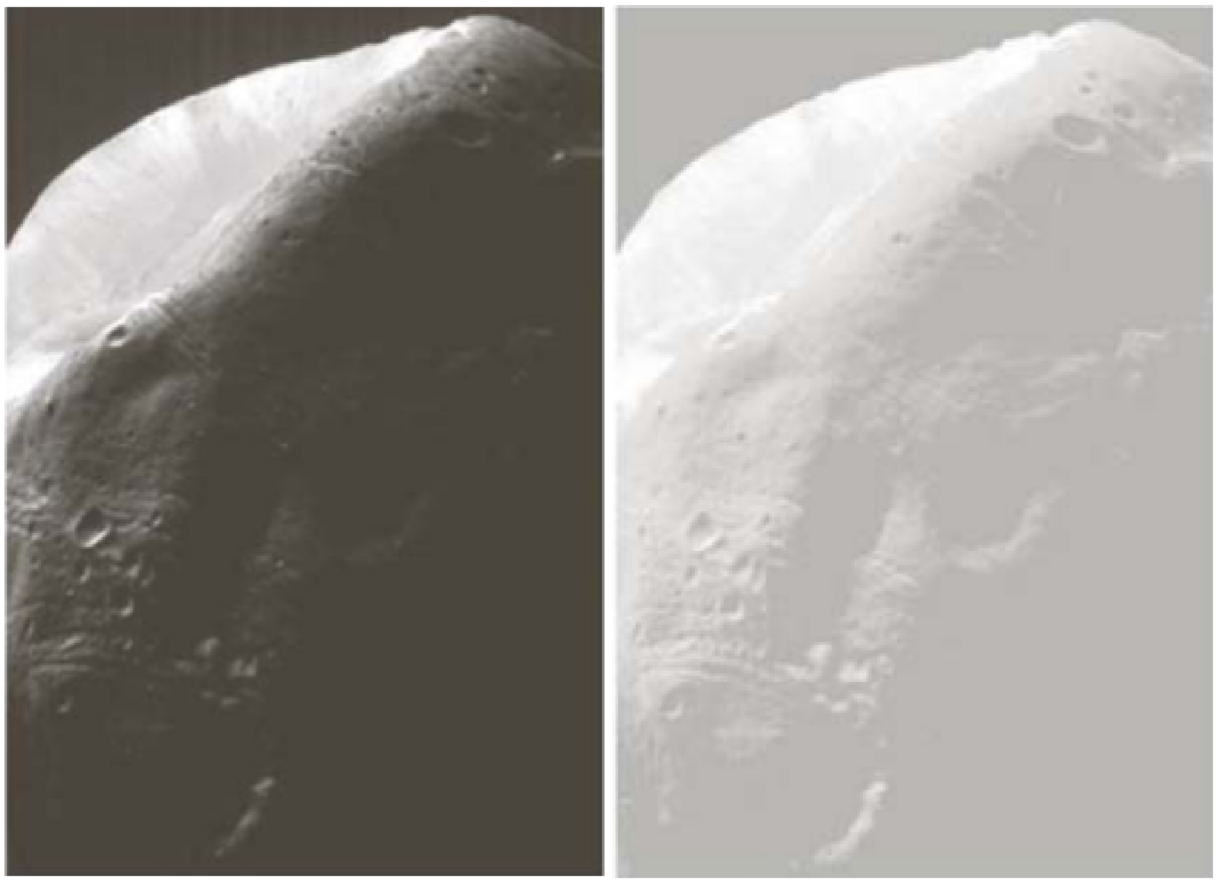


Fig.6 counterexample of histogram equalization

2.2 histogram matching

principle: Histogram matching is also called histogram regularization, which means to adjust the histogram of the image to the specified shape. It matches the histogram of one image to the other image so that the two images are in the same hue. This requires an inverse transformation on the basis of histogram equalization to adjust the uniform shape of the histogram to the specified shape of histogram.

Question formulation: let $p_r(r)$ and $p_z(z)$ denote the continuous probability density functions of the variables r and z. $p_z(z)$ is the specified probability density function.

Let s be the random variable with the property:

$$s = T(r) = (L-1)\int_0^r p_r(r)dr \tag{8}$$

Define a random variable z with the property

$$G(z) = (L-1)\int_0^z p_z(z)dz = s \tag{9}$$

Then

$$z = G^{-1}(s) = G^{-1}[T(r)] \tag{10}$$

For discrete cases, the process is similar to histogram equalization:

$$\begin{gathered} s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \\ G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i) = s_k \\ z_q = G^{-1}(s_k) \end{gathered} \tag{11}$$

However, the operation of the inverse function isn't necessary because the purpose of the inverse function is to find the $G(z_q)$ which is closest to s_k and map r_k to z_q. So, for each item in G_{z_q} , what we just do is to find the s_k which is closest to the item and map r_k to z_q.

pseudo code:

```
1. get the histogram of input_image: $p(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2\ldots\ldots L-1$
```

2. get s: $s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$

3. get $G(z_q)$ according to $p_z(z)$: $G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i)$

4. for each item in G_{z_q}, find the s_k which is closest to the item

5. map r_k to z_q : output_image$[i][j]$=z_q where input_image$[i][j]$=r_k

Python code:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image



def hist_match_12011923(input_image,spec_hist):
    H,W=input_image.shape
    total=H*W*1

    output_image=np.zeros([H,W],dtype=np.uint8)
    input_hist=[]
    output_hist=[]

    #input_histogram
    for i in range(256):
        input_hist.append(np.sum(input_image==i)/(total))

    # calculate s
    r_s_h=0
    r_s=[]
    for i in range(0,256):
        ind=np.where(input_image==i)
        r_s_h+=len(input_image[ind])
        r_s.append(round(255/total*r_s_h))
    #calculate G(z)
    z_s_h=0
    z_s=[]
    for i in range(0,256):
        z_s_h+=spec_hist[i]
        z_s.append(round(255*z_s_h))

    #map r to z
    r_z=[]
    for i in range(256):
        s=r_s[i]
        flag=True
        for j in range(256):
            if z_s[j]==s:
                r_z.append(j)
                flag=False
                break
        if flag==True:
            minp=255
            jmin=0
            for j in range(256):
                b=abs(z_s[j]-s)
                if b<minp:
                    minp=b
                    jmin=j
            r_z.append(jmin)
    #output image
    for i in range(H):
        for j in range(W):
            output_image[i,j]=r_z[input_image[i][j]]

    #output_histtogram
    for i in range(256):
        output_hist.append(np.sum(output_image==i)/(total))

    return(output_image,input_hist,output_hist)

img=cv2.imread('Q3_2.tif',cv2.IMREAD_GRAYSCALE)
cv2.imshow('Q3_2',img)
cv2.waitKey(0)
print(img.shape)
```

```
x=np.arange(256)

img2=cv2.imread('Q3_1_1.tif',cv2.IMREAD_GRAYSCALE)
r,c=img2.shape
spec_hist=[]
for i in range(256):
    spec_hist.append(np.sum(img2==i)/(r*c))

(out,input_hist,output_hist)=hist_match_12011923(img,spec_hist)
plt.imshow(out,cmap=plt.cm.gray)
plt.show()
savedimage1=Image.fromarray(out)
savedimage1.save('Q3_2_hist_match_12011923.tif')
```

In the process of mapping r_k to z_q, for each item in s, we compare it to each item in $G(z_q)$. When finding a item in $G(z_q)$ which is equal to the item in s, the flag becomes $False$, z_q will be put into the appropriate position in $r-z$ and a new cycle will begin. If not, go through each value in $G(z_q)$ to find the closest one, z_q will be put into the appropriate position in $r-z$ and a new cycle will begin. The idea is the same as the above. The remaining process is similar as that in histogram equalization. For the specified probability density function $p_z(z)$, we use the histogram of picture $Q3-1-1.tif$.

result:

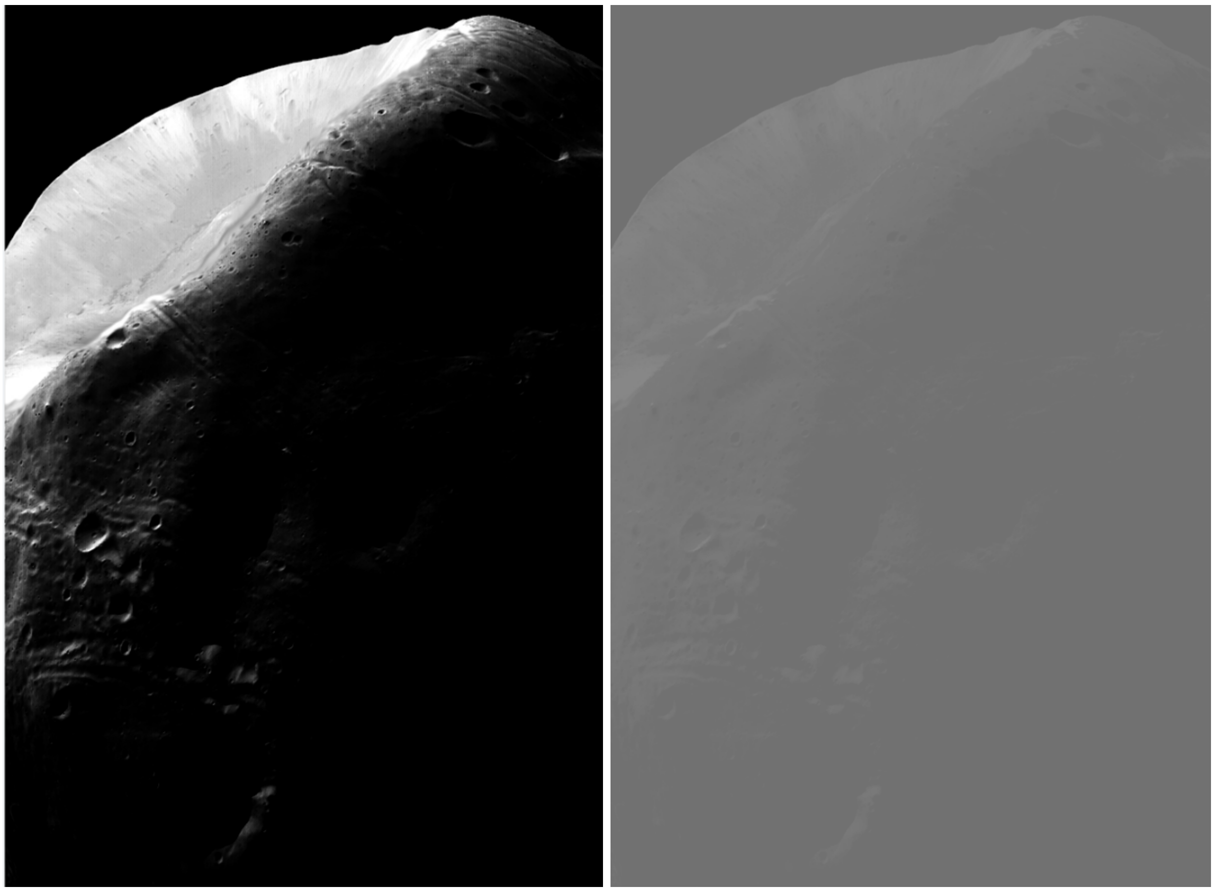


Fig.7 input_image(left) and output_image(right)

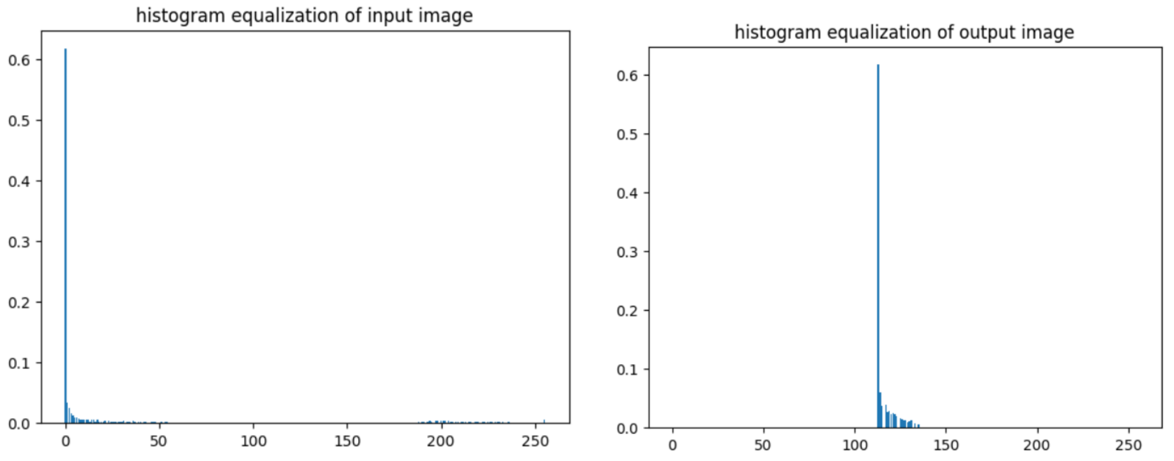


Fig.8 histogram of input_image(left) and histogram of output_image(right)

Analysis: From the figure and histogram of output_image, it is obvious that the output_image and specified_image are in the same hue and the general shape of histogram of input_image is preserved, which proves histogram matching adjust the uniform shape of the histogram to the specified shape of histogram. In this experiment, because the background of specified_image are too bright, the result of histogram matching isn't ideal. When applying histogram in reality, the choice of ideal specified_image is an important step in getting better results.

2.3 local histogram equalization

principle: Histogram equalization and histogram matching are histogram processing for the whole image. Similarly, histogram processing can be used for local enhancement. The step of local histogram is to define a neighborhood and move its center from pixel to pixel. At each location, the histogram of the points in the neighborhood is computed. Either histogram equalization or histogram specification transformation function is obtained. Then, map the intensity of the pixel centered in the neighborhood. After that, move to the next location and repeat the procedure.

Question formulation:

Assume the coordinate of present pixel is $[i, j]$

1. define a neighborhood whose center is the pixel and calculate the histogram of the neighborhood
2. calculate the $s = T(r)$ according to the histogram and map intensity of the pixel centered in the neighborhood
3. move the center point to the next location
4. repeat the above three steps until all pixels have been traversed

Assume the size of input_image is `M×N` and the size of neighborhood is `m_size×m_size`.

pseudo code:

1. Add 0 around the input_image to change the size of the image to `(M+(m_size-1)/2)×(N+(m_size-1)/2)`
2. create loop.
3. In a loop, calculate the $s = T(r)$ according to the histogram and map intensity of the pixel centered in the neighborhood
4. until all pixels have been traversed

Python code:

```python
def local_hist_equ_12011923(input_image,m_size):
    H,W=input_image.shape
    total=H*W*1
    output_image=np.zeros([H,W],dtype=np.uint8)
    input_hist=[]
    output_hist=[]

    #input_histogram
    for i in range(256):
        input_hist.append(np.sum(input_image==i)/(total))

    #往四周补0
    padimage=np.pad(input_image,((int((m_size-1)/2),int((m_size-1)/2)),
(int((m_size-1)/2),int((m_size-1)/2))),'constant',constant_values=(0,0))
    #local histogram processing
    for i in range(int((m_size-1)/2),input_image.shape[0]+int((m_size-1)/2)):
        for j in range(int((m_size-1)/2),input_image.shape[1]+int((m_size-
1)/2)):
            partimage=padimage[i-int((m_size-1)/2):i+int((m_size-1)/2)+1,j-
int((m_size-1)/2):j+int((m_size-1)/2)+1]
            part_hist=np.zeros(256)
            for m in range(partimage.shape[0]):
                for n in range(partimage.shape[1]):
                    part_hist[partimage[m][n]]=part_hist[partimage[m][n]]+1

            r_s=np.cumsum(part_hist[:partimage[int((m_size-1)/2),int((m_size-
1)/2)]+1])/(m_size*m_size)
            output_image[i-int((m_size-1)/2),j-int((m_size-
1)/2)]=int(round(255*r_s[partimage[int((m_size-1)/2),int((m_size-1)/2)]]))


    #output_histogram
    for i in range(256):
        output_hist.append(np.sum(output_image==i)/(total))

    return (output_image,output_hist,input_hist)
```

The parameter, `m_size` is preferably odd because it is helpful to determine the center point of the neighborhood. The function `np.pad` is used to add 0 around the input_image. For better computation and simplicity of code, the size of pad_image should be `(M+(m_size-1)/2)×(N+(m_size-1)/2)`. So, `np.pad(input_image,((int((m_size-1)/2),int((m_size-1)/2)),(int((m_size-1)/2),int((m_size-1)/2))),'constant',constant_values=(0,0))` is used to add 0 to the new locations. Because of the change of size, the interval of row in loops of this pad_image is `[int((m_size-1)/2),input_image.shape[0]+int((m_size-1)/2)]` and the interval of column in loops of this pad_image is `int((m_size1)/2),input_image.shape[1]+int((m_size-1)/2)`, where the input_image is. `partimage=padimage[i-int((m_size-1)/2):i+int((m_size-1)/2)+1,j-int((m_size-1)/2):j+int((m_size-1)/2)+1]` is the neighborhood whose center point is $[i, j]$. When calculating s_k, we just calculate the number of pixels which isn't greater than central pixel according to `formual (7)`, which can reduce the time of computation. The remaining process is similar as that in histogram equalization. Because i and j are the coordinates in the pad_image, `output_image[i-int((m_size-1)/2),j-int((m_size-1)/2)]` is replaced when mapping.

result:

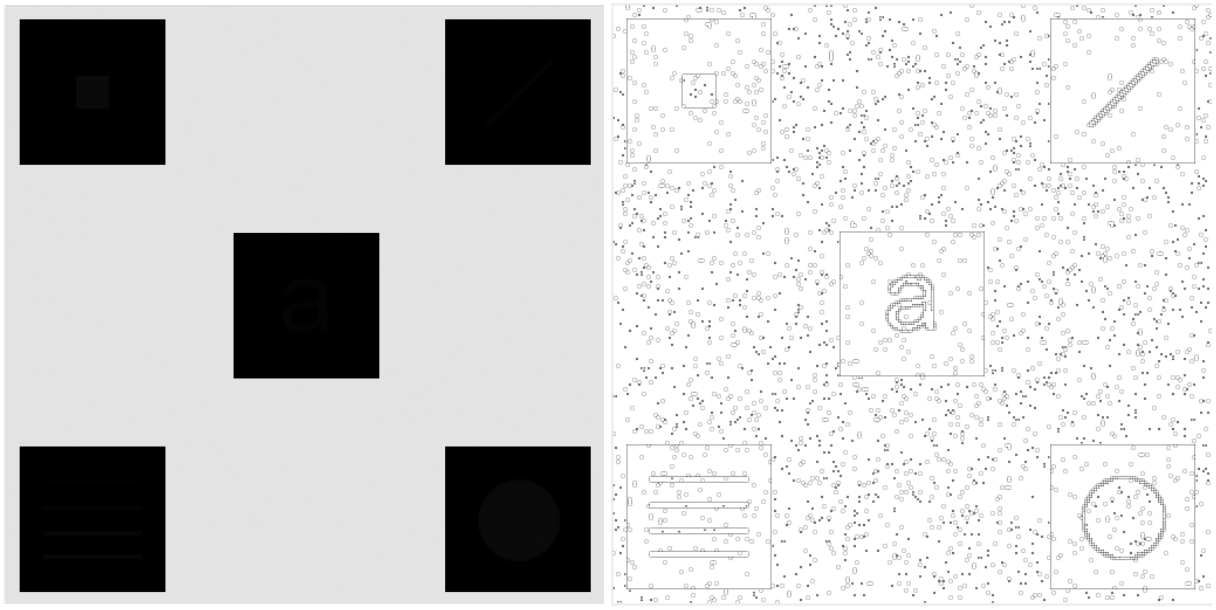


Fig.9 input_image(left) and output_image using local histogram equalization(right)

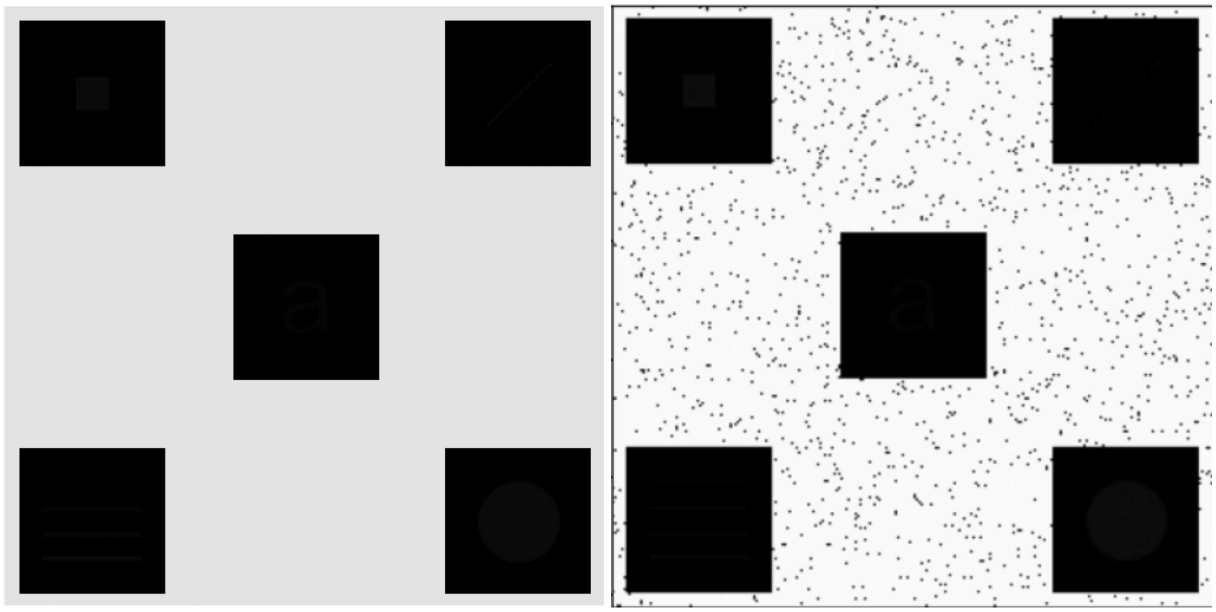


Fig.10 input_image(left) and output_image using global histogram equalization(right)

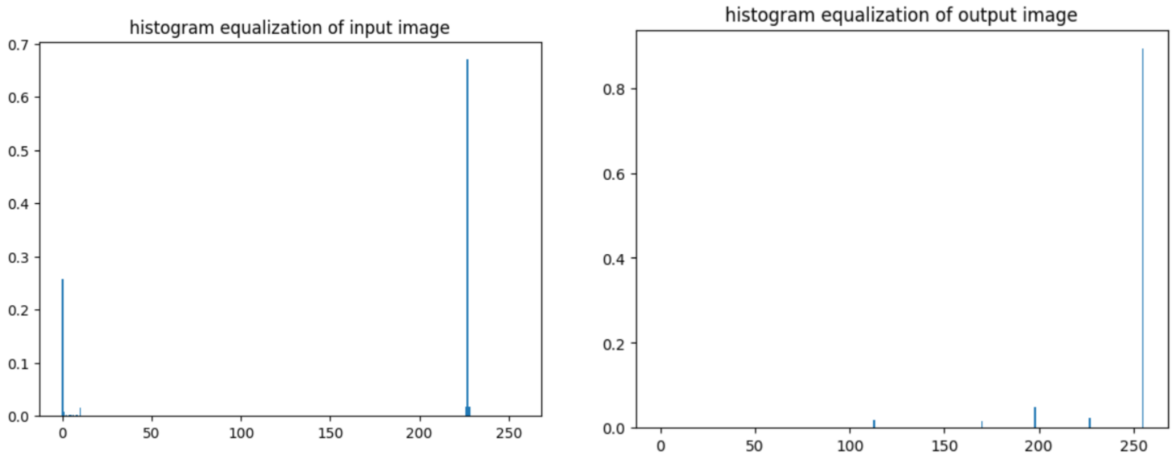


Fig.11 histogram of input_image(left) and histogram of output_image using local histogram equalization(right)

Analysis: The size of neighborhood above is 3×3. By comparing the output_image using local histogram equalization with the output_image using global histogram equalization, it is obvious that the symbols covered by black block are recovered by local histogram equalization. Although there are some blemishes and distortions, this does not affect the message conveyed by the image. Comparing the histogram of input_image and that of output_image, this method not only enhances the contrast of local detail of the image, but also eliminates the effect of pixel with gray value of 0.

2.4 reduce the salt-and-pepper noise

principle: Salt-and-pepper noise is black and white noise generated by image sensors, transmission channels, decoding processes, and so on. Salt-and-pepper noise, also called impulse noise, is the black and white pixels which appear randomly in the image. The most commonly used algorithm for reducing salt-and-pepper noise is median filtering. Median filter is a typical order-statistic filtering technology and it replaces the value of the center pixel with the median value based on ordering the pixels contained in the filter mask, which is similar to local histogram equalization. The basic principle of it is to replace the value of a point in the digital image with the median value based on a neighborhood of the point, which makes the surrounding pixel value is close to the real value so as to eliminate the isolated noise point.

Question formulation:

Assume the coordinate of present pixel is $[i, j]$

1. define a neighborhood whose center is the pixel and order the gray values of points in the neighborhood from smallest to largest
2. map intensity of the pixel centered in the neighborhood to the median value of the ranking result
3. move the center point to the next location
4. repeat the above three steps until all pixels have been traversed

Assume the size of input_image is `M×N` and the size of neighborhood is `n_size×n_size`.

pseudo code:

1. Add 0 around the input_image to change the size of the image to `(M+(n_size-1)/2)×(n+(m_size-1)/2)`
2. create loop.
3. In a loop, order the gray values of points in the neighborhood from smallest to largest and map intensity of the pixel centered in the neighborhood to the median value of the ranking result
4. until all pixels have been traversed

Python code:

```
def reduce_SAP_12011923(input_image,n_size):#n也最好为奇数
```

```
    H,W=input_image.shape
    total=H*W*1

    output_image=np.zeros([H,W],dtype=np.uint8)

    #往四周补0
    padimage=np.pad(input_image,((int((n_size-1)/2),int((n_size-1)/2)),
(int((n_size-1)/2),int((n_size-1)/2))),'constant',constant_values=(0,0))

    for i in range(int((n_size-1)/2),input_image.shape[0]+int((n_size-1)/2)):
        for j in range(int((n_size-1)/2),input_image.shape[1]+int((n_size-
1)/2)):
            partimage=padimage[i-int((n_size-1)/2):i+int((n_size-1)/2)+1,j-
int((n_size-1)/2):j+int((n_size-1)/2)+1]
            #print(partimage)
            #print(np.median(partimage))
            output_image[i-int((n_size-1)/2),j-int((n_size-
1)/2)]=int(round(np.median(partimage)))

    return output_image
```

The parameter, `n_size` is preferably odd because it is helpful to determine the center point of the neighborhood. The function `np.pad` is used to add 0 around the input_image. For better computation and simplicity of code, the size of pad_image should be `(M+(n_size-1)/2)×(N+(n_size-1)/2)`. So, `np.pad(input_image,((int((n_size-1)/2),int((n_size-1)/2)),(int((n_size-1)/2),int((n_size-1)/2))),'constant',constant_values=(0,0))` is used to add 0 to the new locations. Because of the change of size, the interval of row in loops of this pad_image is `[int((n_size-1)/2),input_image.shape[0]+int((n_size-1)/2)]` and the interval of column in loops of this pad_image is `int((n_size1)/2),input_image.shape[1]+int((n_size-1)/2)`, where the input_image is. `partimage=padimage[i-int((n_size-1)/2):i+int((n_size-1)/2)+1,j-int((n_size-1)/2):j+int((n_size-1)/2)+1]` is the neighborhood whose center point is $[i, j]$. The function `np.median` returns the median value of an array. Because i and j are the coordinates in the pad_image, `output_image[i-int((n_size-1)/2),j-int((n_size-1)/2)]` is replaced when mapping.

result:

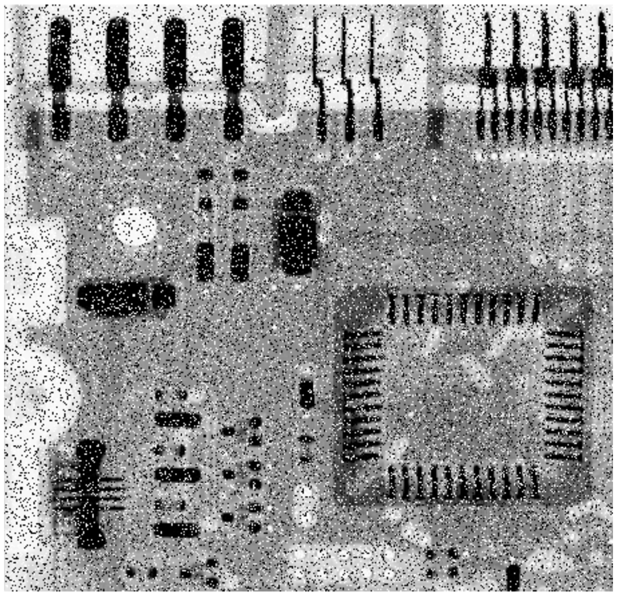


Fig.12 input_image

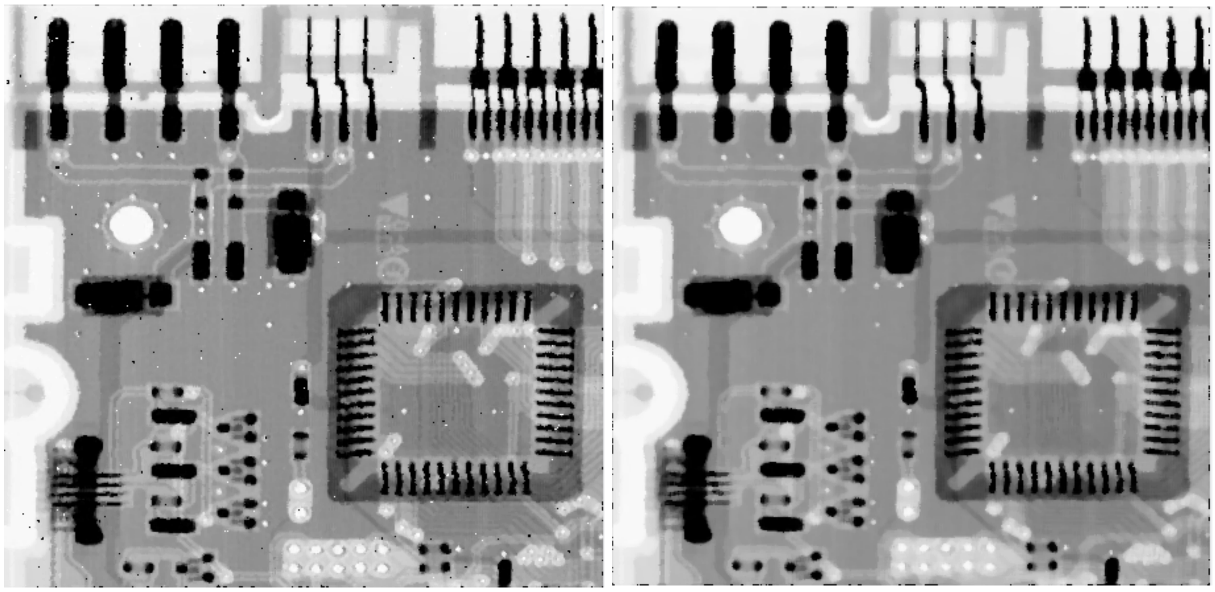


Fig.13 output_image with size 3(left) and size 5(right)

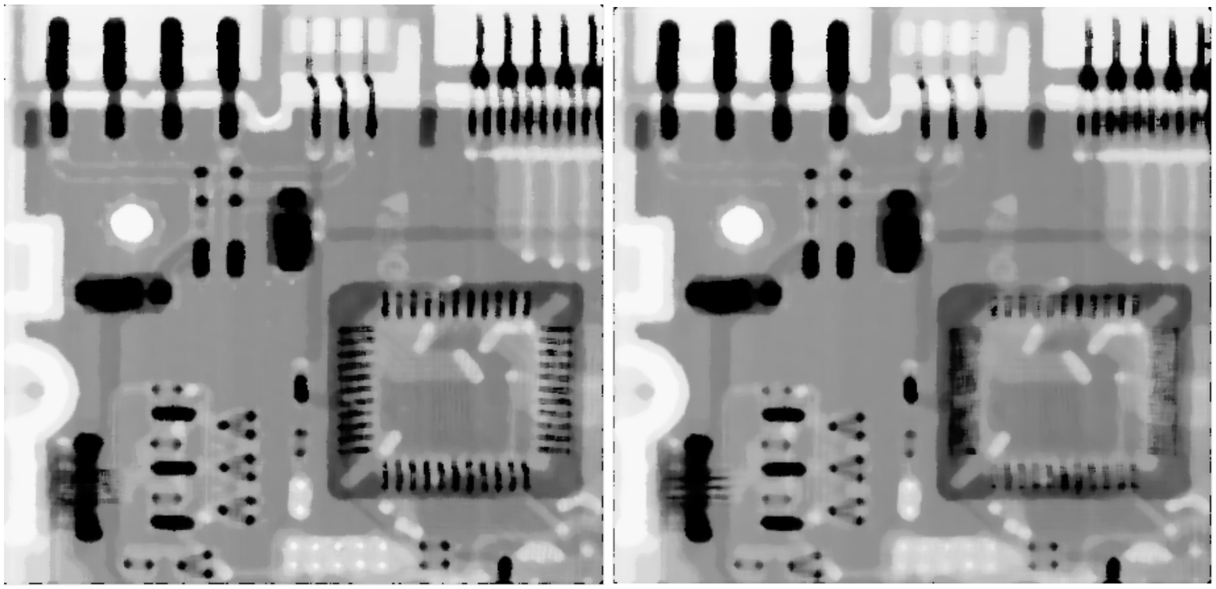


Fig.14 output_image with size 7(left) and size 9(right)

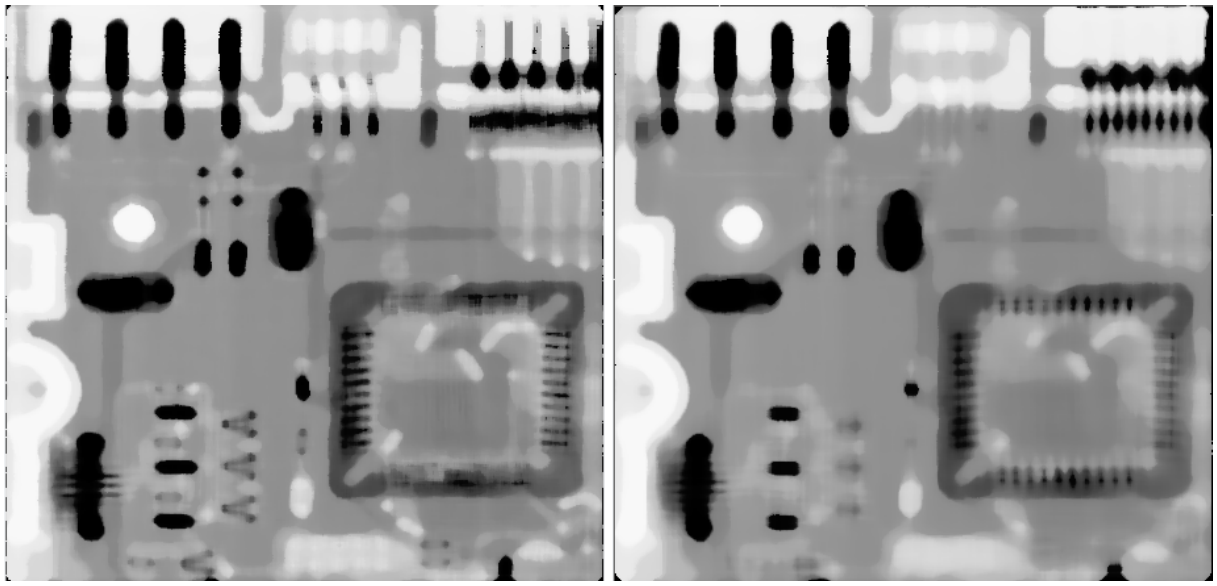


Fig.15 output_image with size 11(left) and size 17(right)

Analysis: From the above pictures, it is obvious that the larger the size of median filter, the better effect of reducing salt-and-pepper noise. However, the image is getting blurry with the size of median filter increasing because the useful information is replaced by the median.

2.5 Comparison of algorithm complexity

The time complexity of an algorithm is one of the important factors affecting the algorithm effect. In this part, we use the `Q3_1_1.tif` as input_images for the four algorithm and the size of neighborhood for local histogram equalization and median filtering is 3×3. Run it ten times and take the average running time. The result is as the following table.

	HISTOGRAM EQUALIZATION	HISTOGRAM MATCHING	LOCAL HISTOGRAM EQUALIZATION	MEDIAN FILTERING
average time/s	0.26	0.92	5.76	5.94

It is concluded that the time complexity of histogram equalization is the lowest among the four algorithms and that of local histogram equalization and median filtering are the highest. What's more, the time complexity of local histogram equalization and median filtering is not only associated with the size of input_image, but also associated with the size of median filtering.

3. Conclusion

From the analysis about the four algorithm, these four algorithms have advantages and disadvantages.

Histogram equalization: The algorithm complexity of histogram equalization is relatively low and it is very useful in images where the background is too bright or too dark. It not only can improve the contrast of the image, but also can transform the image into an image with a nearly uniform distribution of pixel values. It is a fairly straightforward technique and reversible operation. If the transformation function is known, the original histogram can be recovered. Histogram equalization is a global processing method and is indiscriminate in processed data, which may increases the contrast of background interference information and reduce the contrast of useful pixels. The gray level of the image after histogram equalization will be reduced and some details will disappear.

Histogram matching: When the gray level of the image is mostly concentrated in the low level, histogram equalization will cause the gray level of the image to a high level, resulting in a faded image. Histogram matching can preserve the general shape of the image histogram and don't cause the gray level of the image to a high level.

Local histogram equalization: When there is a large change of gray value between bright and dark areas in the image, global histogram equalization can change the overall contrast while most of information are lost in the places where the gray value is high after global equalization. Local histogram equalization can preserve the information in the places where the gray value is high. However, the time complexity of it is high.

Median filtering: It replaces the value of the center pixel with the median value based on ordering the pixels contained in the filter mask to reduce the salt-and-pepper noise. The larger the size of median filter, the better effect of reducing salt-and-pepper noise. However, the image is getting blurry with the size of median filter increasing because the useful information is replaced by the median. And the time complexity of it is high.

In the actual digital image processing process, it is necessary to select the correct algorithm according to the characteristics of the input image and the advantages and disadvantages of the algorithms.