SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF MICROELECTRONIC

Lab 2 Spatial Transforms and Filtering

Student name & ID: *YUAN Tong 11810818*

Course: *LAB Session I* – Professor: *YU Yajun*
Date: *March 14, 2021*

1. Introduction

The term *spatial domain* refers to the image plane itself, and image processing methods in this category are based on direct manipulation of pixels in an image. Here we have two principal categories of spatial processing, intensity transformations and spatial filtering. According to the definition, intensity transformations operate on single pixels of an image, principally for the purpose of contrast manipulation and image threshold, while Spatial filtering deals with performing operations, such as image sharpening, by working in a neighborhood of every pixel in an image.

In the following task, we will apply several classic image processing method to enhance the quality of given images, the methods including histogram equalization, histogram matching, local histogram equalization and de-noising, each methods will be introduced in detail in following sections.

In the current technology field, such kind of algorithm could be applied to improve the quality of photography, to emphasis the details in medical image, and improve the information included in geography photos.

Task 1: histogram equalization

Implement the histogram equalization to the input images Q3_1_1.tif and Q3_1_2.tif.

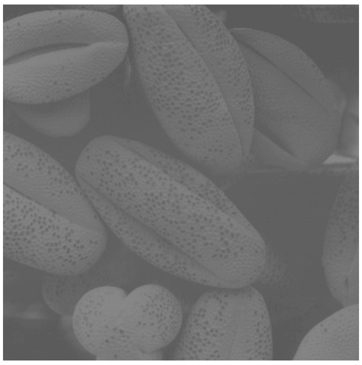


Figure 1: Q3_1_1.tif

Analysis. In histogram equalization, we assume that r is in the range [0, L - 1], with r = 0 representing black and r = L - 1 representing white. For r satisfying these conditions, we define the transformations (intensity mappings) of the form

$$s = T(r) \quad 0 \leqslant r \leqslant L - 1$$

we assume that with the input $0 \leqslant r \leqslant L - 1$ we must obtian $0 \leqslant T(r) \leqslant L - 1$ and $T(r)$ is a strictly monotonically increasing function in the interval $0 \leqslant r \leqslant L - 1$.

Here we define the output of the transformation s with

$$s = T(r)$$

, and the histogram of the image can be defined as its probability density function (PDF), we can obtain the relationship of PDF before and after the transformation:

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|$$

and finally we have

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

for discrete system, we have

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{r} p_r(r_j) = \frac{L-1}{MN}\sum_{j=0}^{r} n_j$$

Here, M and N are the dimension of the image so $M \times N$ is the total number of pixels contained in the image. Then we can apply the method to the given images.

Algorithm 1: Histogram equalization

Calculate the histogram of the input image;
foreach *pixel in input image* **do**
  pixel = $\frac{L-1}{MN}\sum_{j=0}^{r} n_j$
end
Calculate the histogram of the output image;

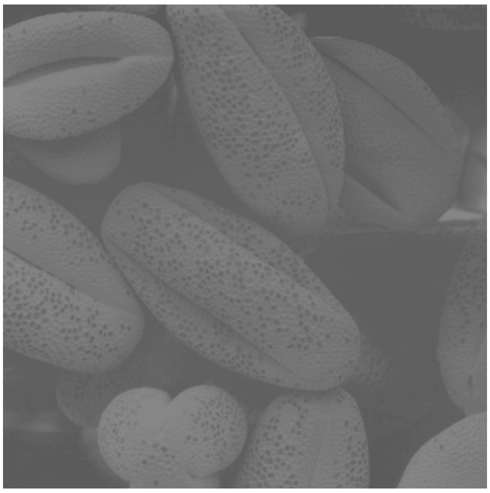


Figure 2: The origin image

Figure 3: The enhanced image

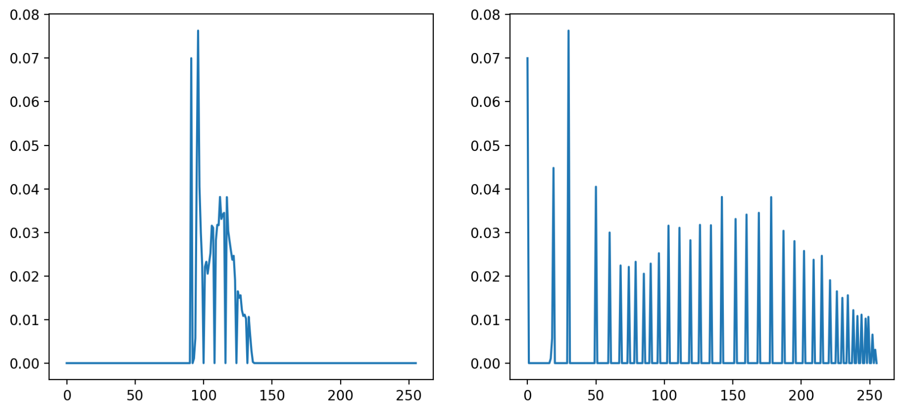


Figure 4: The histograms of input and output image

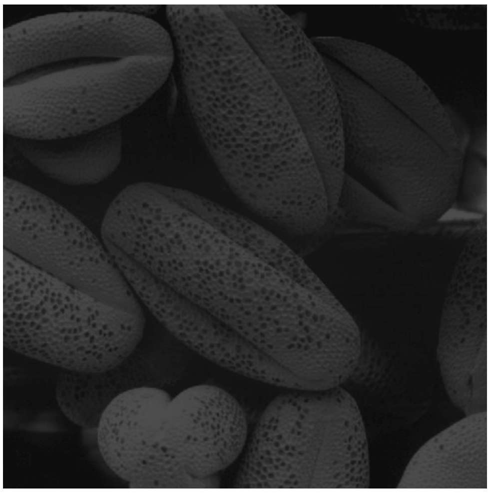


Figure 5: The origin image

Figure 6: The enhanced image

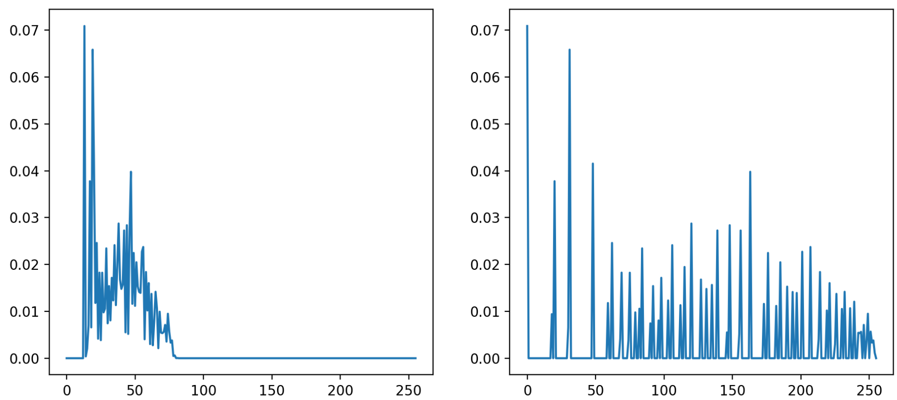


Figure 7: The histograms of input and output image

Result. We find that the given two image have quite different histogram, Figure 2 shows more pixels have middle intensity and Figure 5 shows more in dark region.

After enhancement, the output image look pretty same, but the histogram looks a little different. For example, the output histogram in Figure 7 looks more average, in other words, the intensity of pixels distributed in more level, that's may because the histogram in Fugure 5 cover wider area. If we take a look to output images Figure 3 and Figure 6 we will find that we can't hardly distinguish the difference, but, Figure 6 looks more smooth.

Task 2: histogram matching

Specify a histogram for image Q3_2.tif, such that by matching the histogram of Q3_2.tif to the specified one, the image is enhanced.

Analysis. As indicated in the preceding discussion, histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram. But there are applications in which attempting to base enhancement on a uniform histogram is not the best approach. In particular, it is useful sometimes to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate a processed image that has a specified histogram is called histogram matching or histogram specification.

Here we define z is the output image with the property that

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

with

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

so we can obtain that

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

For this method, we can first apply the histogram equalization to the input image, then, we apply the inverse function $G^{-1}(z)$ to the image after histogram equalization, and the method which is used to handle discrete situation will be discussed later. To make the quality of the image best, we need to select proper p_z for the image.

Algorithm 2: Histogram matching

```
Select proper histogram p_z
Calculate the function G(z)
Derive the inverse function G^{-1}(z)
Calculate the histogram of the input image;
foreach pixel in input image do
    pixel = (L-1)/(MN) ∏_{j=0}^{r} n_j
    output = G^{-1}(pixel)
end
Calculate the histogram of the output image;
```

Figure 8: The origin image

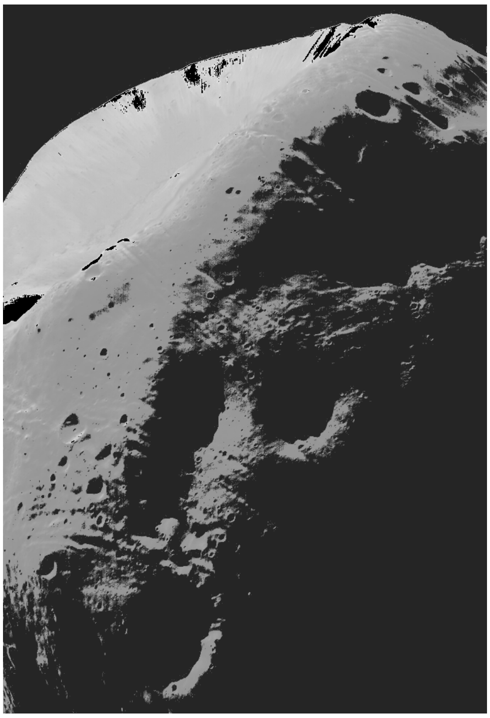


Figure 9: The enhanced image

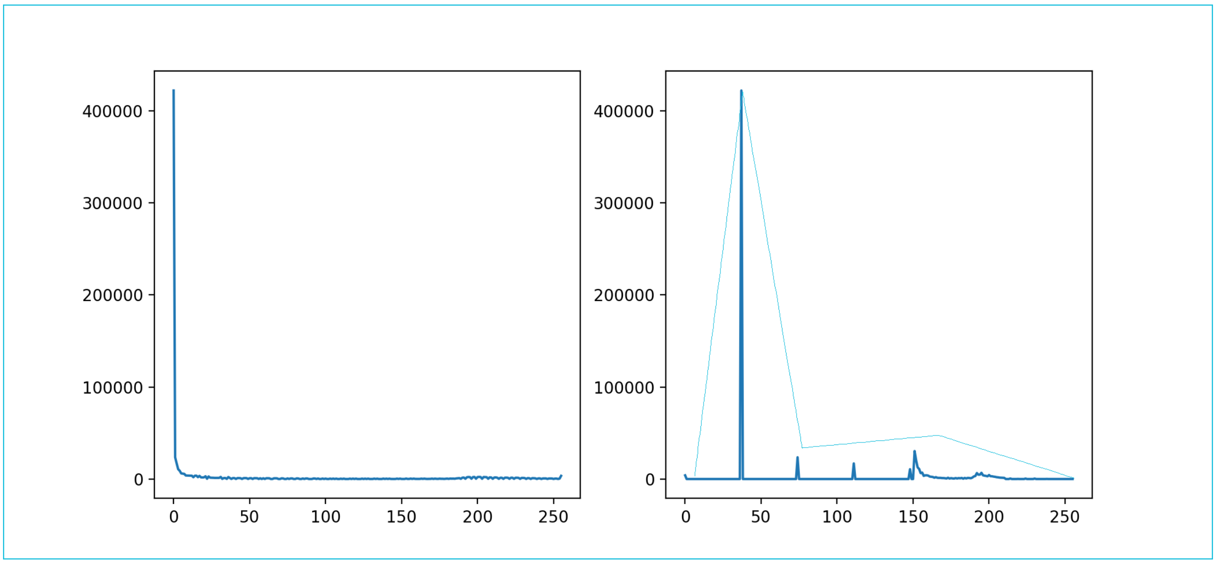


Figure 10: The histogram of the input and output image

Result. In this task we try our best to separate the pixels in different levels, as we can see, after matching with the histogram shown in the plot, the intensity of pixels are accumulated in several peaks, and in the enhanced image, more details are shown.

Discussion.

Comparing. We know that goal of histogram equalization is to produce an output image that has a flattened histogram, the goal of histogram matching is to take an input image and generate an output image that is based upon the shape of a specific (or reference) histogram, also, we can consider histogram equalization as a special case of histogram matching with flat target histogram.

For most of the case, we can enhance the quality of the image by apply histogram equalization to it, however, there are some special situation, if large portion of the image is set to specific color, for example, black and white, the histogram equalization may increase the noise in the area that is totally black or white, in that case we need to design histogram that is suitable for such kind of image.

Discrete situation. As we all know, the image is quantized in computer, so the function $G(z)$ must be discrete, so as the inverse function $G^{-1}(s)$. In this case, when we apply the inverse function $G^{-1}(s)$ to s, there will be some s that is unable to match exactly, here we need to develop methods to solve the problem.

We have know that the function $G_{-1}(s)$ is strictly monotonically increasing.

1. **nearest neighbor:** in the look up table of function $G^{-1}(s)$, we will find the closest number and use it.
2. **linear:** in the table, find the two values at the two side, and apply the linear interpolation to fill the value.

But as we tested, we could not distinguish the difference between two method, so to decrease the time consuming of the program, we choose to use nearest neighbor interpolation to solve the problem.

Specified histogram. There are many kind of different histograms that can be used to match the given image. In Figure 11 the image was matched by uniform distributed histogram, and we find that still lots of details are missing. And in Figure 12 we lighten up the whole image, and details can not be identified too.

Figure 11: Enhanced by histogram equalization

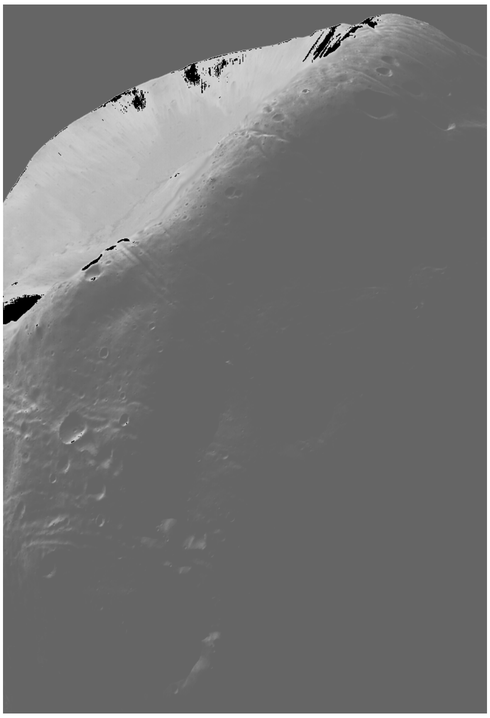


Figure 12: Matched by another histogram diagram

Task 3: local histogram equalization

Implement the local histogram equalization to the input images Q3_3.tif.

Analysis. Despite two cases analysed before, there are cases in which it is necessary to **enhance details over small areas in an image.** The number of pixels in these areas may have negligible influence on the computation of a global transformation whose shape does not necessarily guarantee the desired local enhancement. The solution is to devise transformation functions based on the intensity distribution in a neighborhood of every pixel in the image. The procedure of this method 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 and either a histogram equalization or histogram specification transformation function is obtained. This function is then used to map the intensity of the pixel centered in the neighborhood.The center of the neighborhood region is then moved to an adjacent pixel location and the procedure is repeated. Because only one row or column of the neighborhood changes during a pixel-to-pixel translation of the neighborhood, updating the histogram obtained in the previous location with the new data introduced at each motion step is possible (Problem 3.12). This approach has obvious advantages over repeatedly computing the histogram of all pixels in the neighborhood region each time the region is moved one pixel location. Another approach used sometimes to reduce computation is to utilize nonoverlapping regions, but this method usually produces an undesirable "blocky" effect.

Algorithm 3: Local histogram equalization

Calculate the histogram of the input image;
foreach *pixel in input image* **do**
  extract the $m \times m$ neighbor
  apply histogram equalization to the neighbor
end
Calculate the histogram of the output image;

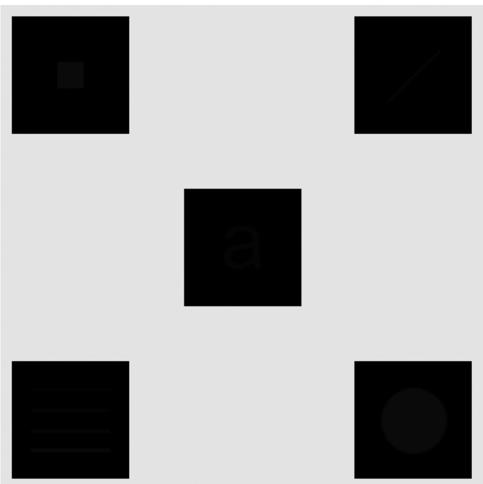


Figure 13: The origin image

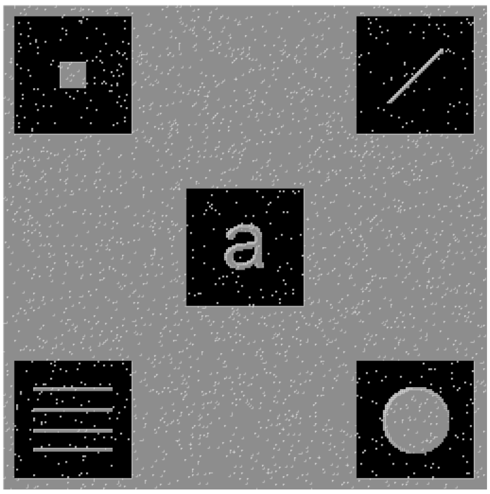
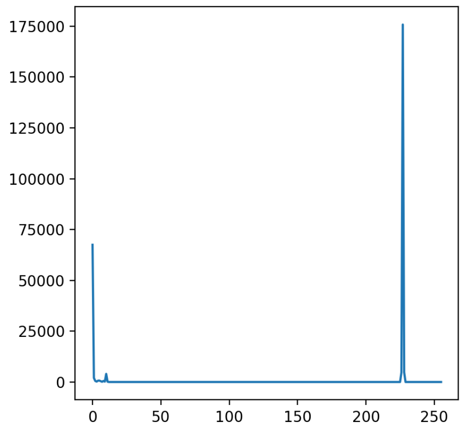


Figure 14: The enhanced image



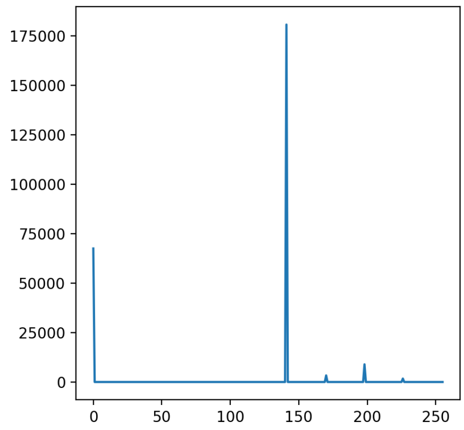


Figure 15: Histogram of input and output image

Result. In this task, we apply local histogram equalization to Figure 13, and we find that in the output image 14, several changes occurred.

1. The pixels with high intensity (close to white) received a decrease in intensity (close to black), while pixels close to black remain almost unchanged. In Figure 15 we could infer that the highest peak, which represents the gray pixels in Figure 13, has moved lefty.

2. The noise is increased. In Figure 15 we can find in input histogram that some noise appeared near the second highest peak. And in output histogram, more noise appeared near the highest peak.

3. some information contained in dark zone is displayed.

Discussion. In this task, the method of edge pending and the size of the neighborhood will affect the output, although we may unable to identify the difference contains in the output images, the methods will be discussed here.

Pending. Many methods are applied to solve the problem that when the neighbor can't be filled at edge and corner, here several method is listed here.

1. Fill the black pixels with number 0 or 1

2. Fill the black pixels with the nearest pixle.

3. Fill the black pixels by bicubic interpolation.

As a matter of fact, if we do not care the details in several lines at the edge, we can just fill the blank pixels with its nearest pixel.

size. And also the size of the neighbor will affect the result of enhancement, generally speaking, the size of the neighbor will affect that how many pixels will be involved when we apply the enhancement for each pixel. For example, if we apply larger m, at the edge there will be a smooth decrease rather than sharp change of the color.

Task 4: filter

Implement an algorithm to reduce the salt-and-pepper noise of an image. The input image is Q3_4.tif.

Analysis. Order-statistic filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.

The best-known filter in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the intensity values in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median).

Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.

Algorithm 4: Median filter

foreach *pixel in input image* **do**
  extract the $m \times m$ neighbor
  sort the intensity of the pixels in the neighbor region.
  choose the middle intensity pixels and replace the center pixel by it.
end
Calculate the histogram of the output image;

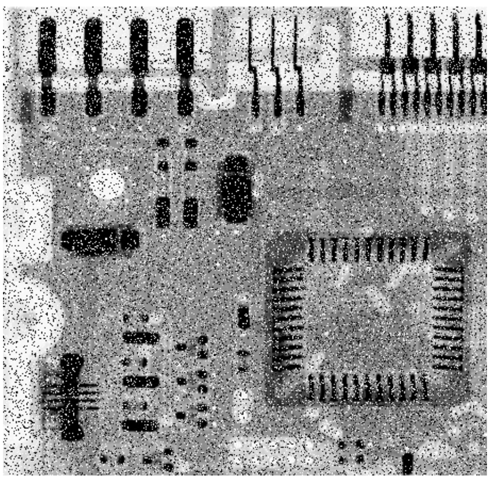


Figure 16: The origin image

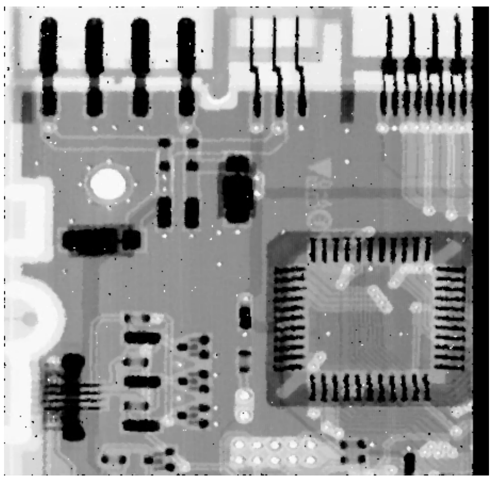


Figure 17: The enhanced image

Result. In figure 16, we can found a lot of salt-and-petty noise inside it, a significant property of this kind of noise is that all the noise is at the highest pixel or the lowest pixel, so here we can use median filter to enhance the image. As it shown in Figure 17, most of the noise is removed, but it seems that the enhanced image is too smooth, that's because most of the change of pixels are also removed.

Discussion. In the result we still found two problems, one is that the image is too smooth that some details is lost, and also some noise remained in the enhanced image, we should modify some details in the program.

1. there might be some noise pixels connect to each other so it may be unable to be removed, we can apply the algorithm twice to remove more noise.
2. Also, we find that all the noise pixels have intensity close to 0 or 255, so we can just pick up them an do the demonising rather than filet each pixels.

Conclusion

In this lab we applied different kinds of method for image enhancement, we find that different method can enhance different kind of image.

1. For normal image, we can just apply histogram equalization, and we can obtain an enhanced image with all details clear.
2. For image with some pixels occupied a large portion but shouldn't be adjusted, we could use histogram matching to specify the target histogram.
3. For those image care more on detail rather than color, we could use local histogram to enhance it, so we can emphasis its details.

And also, for image with specific noise, we can use particular filter to solve it, For example here we use medium filter to de-noise image with salt-and-popper noise.

To draw a conclusion, **the enhancement of an image does not always indicating that to let the image looks more beautiful, the purpose of image enhancement is to make the image more useful**. So for some special purpose, we need to make the details more clear.

Source code

```
"""
LAB 3 Task I:

Implement the histogram equalization to the input images Q3_1_1.tif and Q3_1_2.tif.
"""

import numpy as np
from skimage import io, data
import math
from scipy import interpolate
import matplotlib.pyplot as plt


def sum(histogram, index):
    sum = 0
    for i in range(index):
        sum = sum + histogram[i]
    # print(sum)
    return sum

def hist_equ_11810818(input_image):

    # Define outputs
    output_image = np.zeros(input_image.shape, dtype=np.uint8)

    m,n = input_image.shape

    number_of_pixel = m * n

    input_hist = []  # Distribution of input pixels
    output_hist = []  # Distribution of output pixels

    # Count input
    for i in range(256):
        input_hist.append(np.sum(input_image == i)/number_of_pixel)
    # print(input_hist)

    # histogram equalization
    for i in range(m):
        for j in range(n):
            output_image[i, j] = ((256-1))*sum(input_hist, input_image[i, j])

    # Count output
```

```python
    for i in range(256):
        output_hist.append(np.sum(output_image == i)/number_of_pixel)


    return (output_image, output_hist, input_hist)

if __name__ == '__main__':
# Image 1

    # Process image
    [output_image_1, output_hist_1, input_hist_1] =
    ↪  hist_equ_11810818(io.imread("Q3_1_1.tif"))

    # Print result
    io.imsave("Q3_1_1_11810818.tif", output_image_1)

    # Plot histogram
    fig1, [in_1, out_1] = plt.subplots(1, 2)
    in_1.plot(np.arange(256), input_hist_1)
    out_1.plot(np.arange(256), output_hist_1)

# Image 2

    # Process image
    [output_image_2, output_hist_2, input_hist_2] =
    ↪  hist_equ_11810818(io.imread("Q3_1_2.tif"))

    # Print result
    io.imsave("Q3_1_2_11810818.tif", output_image_2)

    # Plot histogram
    fig2, [in_2, out_2] = plt.subplots(1, 2)
    in_2.plot(np.arange(256), input_hist_2)
    out_2.plot(np.arange(256), output_hist_2)

    plt.show()


```

```python
"""
LAB 3 Task II:

Specify a histogram for image Q3_2.tif.
"""

import numpy as np
```

```python
from skimage import io, data
import math
import matplotlib.pyplot as plt

def sum(histogram, index):
    sum = 0
    for i in range(index):
        sum += histogram[i]
    # print(sum)
    return sum

def match(histogram, pixel):
    for i in range(histogram.shape[0]):
        if pixel < histogram[i]:
            return i

    return 0;

def spec_hist_1():
    spec_hist = np.zeros(256)

    spec_image = io.imread("Q3_2_spec.png")

    for i in range(256):
        spec_hist[i] = (np.sum(spec_image == i))

    spec_hist = spec_hist/np.sum(spec_hist)

    figure, ax = plt.subplots()
    ax.plot(np.range(256), spec_hist)
    plt.show()

    return spec_hist

def spec_hist_2():
    spec_hist = np.zeros(256)

    for i in range(20):
        spec_hist[i] = 0
    for i in range(20, 100):
        spec_hist[i] = 10*i
    for i in range(100, 210):
        spec_hist[i] = 2550 - 10*i
    for i in range(210, 256):
        spec_hist[i] = 256-i

    figure, ax = plt.subplots()
```

```
    ax.plot(range(256), spec_hist)
    plt.show()

    spec_hist = spec_hist/np.sum(spec_hist)
    return spec_hist

def spec_hist():
    spec_hist = np.zeros(256)
    for i in range(100):
        spec_hist[i] = 0
    for i in range(100, 210):
        spec_hist[i] = 25500 - 10*i
    for i in range(210, 256):
        spec_hist[i] = 256-i

    spec_hist = spec_hist/np.sum(spec_hist)

    figure, ax = plt.subplots()
    ax.plot(range(256), spec_hist)
    plt.show()

    return spec_hist

def hist_match_11810818(input_image, spec_hist):

# Define outputs
    output_image = np.zeros(input_image.shape, dtype=np.uint8)

    m,n = input_image.shape

    number_of_pixel = m * n

    input_hist = []  # Distribution of input pixels
    output_hist = []  # Distribution of output pixels

    # Count input
    for i in range(256):
        input_hist.append(np.sum(input_image == i))
    # print(input_hist)

    # histogram equalization
    for i in range(m):
        for j in range(n):
            output_image[i, j] =
             ↪  ((256-1)/number_of_pixel)*sum(input_hist, input_image[i,
             ↪  j])

```

```

    # histogram matching
    G_z = np.zeros((256), dtype=np.uint8)


    for i in range(256):
        # print((256-1)*sum(spec_hist, i))
        G_z[i] = (256-1)*sum(spec_hist, i)
    print(G_z)

    for i in range(m):
        for j in range(n):
            output_image[i, j] = match(G_z, input_image[i, j])

    # Count output
    for i in range(256):
        output_hist.append(np.sum(output_image == i))


    return (output_image, output_hist, input_hist)


if __name__ == '__main__':


    [output_image_1, output_hist_1, input_hist_1] =
     ↪  hist_match_11810818(io.imread("Q3_2.tif"), spec_hist())

    # Print result
    io.imsave("Q3_2_11810818.tif", output_image_1)

    # Plot histogram
    fig1, [in_1, out_1] = plt.subplots(1, 2)
    in_1.plot(np.arange(256), input_hist_1)
    out_1.plot(np.arange(256), output_hist_1)

    plt.show()
```

```
"""
LAB 3 Task III:

Implement the local histogram equalization to the input images Q3_3.tif.
"""

import numpy as np
from skimage import io, data
```

```
import math
import matplotlib.pyplot as plt

def extract_lacal(input_image, x, y, m_size):
    step = int((m_size-1)/2)
    local = np.zeros((m_size, m_size), dtype=np.uint8)

    for i in range(x - step, x + step):
        for j in range(y - step, y + step):
            if i >= 0 and i < input_image.shape[0] and j >= 0 and j <
             ↪ input_image.shape[0]:
                local[i - (x - step), j - (y - step)] = input_image[i,
                 ↪ j]

    return local

def hist_equ(local):

    number_of_pixel = local.shape[0]*local.shape[1]

    center_x = int((local.shape[0]-1)/2)
    center_y = int((local.shape[1]-1)/2)

    input_hist = []  # Distribution of input pixels
    # Count input
    for i in range(256):
        input_hist.append(np.sum(local == i))

    output = ((256-1)/number_of_pixel)*sum(input_hist, local[center_x,
     ↪ center_y])

    return output

def sum(histogram, index):
    sum = 0
    for i in range(index):
        sum = sum + histogram[i]
    # print(sum)
    return sum

def local_hist_equ_11810818(input_image, m_size):

    output_image = np.zeros(input_image.shape, dtype=np.uint8)

    m,n = input_image.shape
    number_of_pixel = m * n

```

```
    input_hist = []  # Distribution of input pixels
    output_hist = []  # Distribution of output pixels

    # Count input
    for i in range(256):
        input_hist.append(np.sum(input_image == i))
    # print(input_hist)

    # local histogram equalization
    for i in range(m):
        for j in range(n):
            print("(" + str(i)+", "+str(j)+")")
            local = extract_lacal(input_image, i, j, m_size)
            output_image[i, j] = hist_equ(local)

    # Count output
    for i in range(256):
        output_hist.append(np.sum(output_image == i))


# Insert code here
    return (output_image, output_hist, input_hist)


if __name__ == '__main__':

    [output_image_1, output_hist_1, input_hist_1] =
     ↪  local_hist_equ_11810818(io.imread("Q3_3.tif"), 3)

    # Print result
    io.imsave("Q3_3_11810818.tif", output_image_1)

    # Plot histogram
    fig1, [in_1, out_1] = plt.subplots(1, 2)
    in_1.plot(np.arange(256), input_hist_1)
    out_1.plot(np.arange(256), output_hist_1)
```

```
"""
LAB 3 Task IV:

"""

import numpy as np
from skimage import io, data
import math
import matplotlib.pyplot as plt
```

```python

def reduce_SAP_11810818(input_image, n_size):

    output_image = np.zeros(input_image.shape, dtype=np.uint8)

    m,n = input_image.shape
    number_of_pixel = m * n

    for i in range(m):
        for j in range(n):
            step = (int)((n_size-1)/2)
            pixels = np.zeros(n_size*n_size)

            for i2 in range(n_size):
                for j2 in range(n_size):
                    if i-step+i2 >= 0 and i-step+i2 < input_image.shape[0] and j-step+j2 >= 0 and j-step+j2 < input_image.shape[0]:
                        pixels[j2*n_size+i2] = input_image[i-step+i2, j-step+j2]

            # print(pixels)
            pixels = np.sort(pixels, axis=None)
            # print(pixels)

            output_image[i, j] = pixels[(int)((n_size*n_size-1)/2)]


    return output_image

if __name__ == '__main__':

    output_image_1 = reduce_SAP_11810818(io.imread("Q3_4.tif"), 3)

    # Print result
    io.imsave("Q3_4_11810818.tif", output_image_1)
```