Laboratory 3: Spatial Transforms and Filtering

Jincheng Yu

March 14, 2023

1 Introduction

Spatial transforms and filtering are fundamental concepts in digital image processing, with numerous practical applications in areas such as computer vision, medical imaging, remote sensing, and multimedia. The goal of this experiment is to implement and evaluate various image processing techniques based on spatial transforms and filtering.

In this experiment, we have implemented several methods, including histogram equalization, histogram matching, local histogram equalization, and salt-and-pepper noise removal. These methods were implemented using Python programming language, and their effectiveness was tested by processing images. Histogram equalization is a widely used technique for enhancing image contrast by redistributing the intensity values of an image. Histogram matching, on the other hand, is a method for matching the histogram of a target image to that of a reference image. Local histogram equalization is a variation of histogram equalization that applies the method to small regions of an image rather than the entire image. Salt-and-pepper noise is a common type of image corruption that results in random black and white pixels in an image, and its removal can improve the visual quality of an image.

In this report, we will discuss the implementation of the aforementioned image processing techniques and their effectiveness in enhancing image quality. The results obtained from this experiment will provide insights into the practical applications of spatial transforms and filtering techniques in digital image processing.

2 Spatial Transforms and Filtering Methods

2.1 Histogram Equalization

Histogram equalization maximizes image information content by mapping pixel intensities to a new range that spans the entire intensity range, resulting in a uniform cumulative distribution function and enhanced image contrast. Let the input image be denoted by $f(x, y)$, where x and y are the spatial coordinates of the image. The histogram of the image is a plot of the frequency of occurrence of each possible intensity value in the image. Let $h(i)$ be the histogram of the input image, where i denotes the intensity level. The CDF $H(i)$ of the image is defined as the sum of the histogram values up to and including intensity level i:

$$H(i) = \sum_{j=0}^{i} h(j)$$

The goal of histogram equalization is to find a mapping function T that transforms the input image $f(x, y)$ into a new image $g(x, y)$ such that the CDF of $g(x, y)$ is a linear function of the intensity levels. The mapping function T is defined as:

$$T(i) = \frac{L-1}{M \times N} \sum_{j=0}^{i} h(j)$$

where L is the total number of possible intensity values, and M and N are the dimensions of the image. The transformed image $g(x, y)$ is obtained by applying the mapping function T to each pixel in the input image $f(x, y)$:

$$g(x, y) = T(f(x, y))$$

The resulting histogram of the transformed image is approximately uniform, which means that the image has maximum contrast.

2.2 Histogram Matching

Histogram matching adjusts the pixel intensities of an image to match a target histogram, often used in image processing for enhancement, color correction, and registration.The basic idea behind histogram matching is to map the pixel values in the input image to new values that correspond to the desired target histogram. This can be done by computing the cumulative distribution function (CDF) of both the input image and the target histogram, and then using these functions to map the pixel values. Let $f(x)$ be the probability density function (PDF) of the input image, and $g(z)$ be the PDF of the target histogram. The CDF of the input image is defined as:

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

Similarly, the CDF of the target histogram is:

$$G(z) = \int_{-\infty}^{z} g(t)dt$$

To perform histogram matching, we need to find a mapping function T that maps the pixel values r in the input image to new values s such that $s = T(r)$ and the resulting image has a histogram that matches the target histogram. The mapping function T can be computed as:

$$T(r) = G^{-1}(F(r))$$

where G^{-1} is the inverse of the CDF of the target histogram. In other words, we find the pixel value s in the target histogram that corresponds to the CDF value of the pixel value r in the input image.

2.3 Local Histogram Equalization

Local Histogram Equalization is a algorithm used to enhance the contrast of an image by performing histogram equalization on smaller regions of the image instead of the entire image. This approach is especially useful for images that have regions with varying illumination levels.
It works by dividing the image into non-overlapping blocks of size N x N pixels. For each block, a histogram of pixel intensities is computed, and then the histogram is equalized using the following formula:

$$p_i(k) = \frac{n_i(k)}{N^2} \tag{1}$$

where $p_i(k)$ is the normalized probability of pixel intensity k in block i, $n_i(k)$ is the number of pixels in block i with intensity k, and N^2 is the total number of pixels in block i.
The cumulative distribution function of the histogram is then computed as follows:

$$C_i(k) = \sum_{j=0}^{k} p_i(j) \tag{2}$$

Finally, the pixel values in the block are transformed using the following equation:

$$g(x, y) = \frac{L-1}{N^2} \sum_{k=0}^{f(x,y)} C_i(k) \tag{3}$$

where $g(x, y)$ is the transformed pixel value at location (x, y), L is the maximum possible pixel intensity value, and $f(x, y)$ is the pixel intensity at location (x, y).
The transformed pixel value $g(x, y)$ is then assigned to the corresponding pixel location in the output image.

2.4 Salt-and-pepper Noise Removal

Median filter is a commonly used method to remove salt-and-pepper noise from images. Assuming we have an image I with salt-and-pepper noise, we first define a square window of size k $\times$ k, where k is an odd integer. We then slide the window over the image, and for each window position, we replace the center pixel value with the median of the pixel values within the window. This operation is repeated for each pixel in the image. The median filter is a non-linear filter, as it replaces a pixel value with the median value within the window.
Mathematically, the output of the median filter, denoted by I', can be defined as follows:

$$I'(x, y) = \text{median} I(x + i, y + j), \text{for } i, j \in [-\frac{k-1}{2}, \frac{k-1}{2}] \tag{4}$$

where I(x,y) is the pixel value at position (x,y) in the original image I, and I'(x,y) is the corresponding pixel value in the filtered image I'. The median operator computes the median value of the pixel values within the k × k window centered at position (x,y).
The median filter is effective in removing salt-and-pepper noise because it replaces the noisy pixel values with the median value of the surrounding pixels, which is less sensitive to outliers than the mean value. Furthermore, the median filter does not blur the edges or details of the image as much as linear filters like the mean filter.

3 Experiment

3.1 Implement

3.1.1 Histogram Equation

Histogram need to calculate new grayscale of every pixel in output according to the mapping function. The algorithm is as following.

```
  Data: Input image $I$
  Result: Output image $J$; Input image histogram $H_I$; Output image histogram $H_J$
1 Calculate the grayscale histogram $H_I$;
2 Calculate the mapping function $T$;
3 Initialize the output image $J$;
4 for every pixel $(i,j)$ in $I$ do
5     Calculate new grayscale $J(i,j) = T(I(i,j))$ at $(i,j)$;
6 end
7 Convert matrix to image;
8 Calculate the grayscale histogram $H_J$;
```

In the python, the following code is used to calculate the histogram of an image array.

```python
for line in arr:
        for point in line:
            input_hist[point] += 1
   N = width * height
   pr = input_hist / N
```

Based on the definition of CDF, it can be obtained by cumulatively summing the histogram, which can be generated using the numpy function cumsum().

```python
Trans = np.int32(255 * np.cumsum(pr))
```

By applying the mapping function to each point and performing calculations, we can generate the output array.

```python
 for i in range(0, width):
        for j in range(0, height):
            output_arr[i, j] = Trans[arr[i, j]]
```

3.1.2 Histogram Matching

The histogram matching calculates the reflection between the histogram of input image and specific histogram and the new grayscale of every point in output figure according to the reflection. The algorithm is shown as follows.

```
Data: Input image $I$; Specific histogram $H$
Result: Output image $J$; Input image histogram $H_I$; Output image histogram $H_J$
1 Calculate the grayscale histogram of input image $H_I$;
2 Calculate the CDF of them $F(x)$ and $G(z)$;
3 Calculate mapping function $f$:$f(i) = \text{round}\left(255 \cdot \sum_{j=0}^{i} H(j)\right)$;
4 for every pixel $(i, j)$ in $I$ do
5     Calculate new grayscale $J(x, y) = f(I(x, y))$ at $(i, j)$;
6 end
7 Convert matrix to image;
8 Calculate the grayscale histogram of input image $H_J$;
```

Based on the process of histogram equation, we have encapsulated the process of calculating a histogram into a function. This allows us to generate a desired histogram for any image by utilizing the function.

```python
def array_to_hist(input_image):
    arr = np.array(input_image).astype(np.int32)
    [width, height] = arr.shape
    output_hist = np.zeros(256)
    for line in arr:
        for point in line:
            output_hist[point] += 1
    N = width * height
    output_hist = output_hist / N
    return output_hist
```

The mapping function is based on the nearest value matching which means we establish the reflection of the similar values between input image and output image.

```python
    idx = 0
    for i in range(256):
        minv = 1
        for j in hist_after[idx:]:
            if np.fabs(hist_after[j] - hist_before[i]) <= minv:
                minv = np.fabs(hist_after[j] - hist_before[i])
                idx = int(j)
        M[i] = idx
```

3.1.3 Local Histogram Equalization

The algorithm divides the histogram equation algorithm into segments and process the subarray of fixed range with histogram equation. The algorithm shows as following:

```
Data: Input image $I$; Specific size $m$
Result: Output image $J$; Input image histogram $H_I$; Output image histogram $H_J$
1  for Every segment $S = I(i : i + m, j : j + m)$ in $I$ do
2      Calculate the grayscale histogram $H_S$;
3      Calculate the mapping function $T$;
4      Initialize the output image $P$;
5      for Every point $(i, j)$ in $S$ do
6          Calculate new grayscale $P(i, j) = T(S(i, j))$ at $(i, j)$;
7      end
8  end
9  Convert matrix to image;
10 Calculate the grayscale histogram $H_J$;
```

Based on the specific size, we can calculate the segments to utilize histogram equation.

```python
    height, width = arr.shape
    seg_height, seg_width = m_size, m_size
    rows = height // seg_height
    cols = width // seg_width
```

The histogram equation has been encapsulated into the function hist_equ(). By calculating all segments of the image, we can obtain the output image.

```python
    for i in range(rows):
        for j in range(cols):
            y = i * seg_height
            x = j * seg_width
            seg = arr[y:y + seg_height, x:x + seg_width]
            output_arr[y:y + seg_height, x:x + seg_width] = hist_equ(
                seg)
```

3.1.4 Salt-and-pepper Noise Removal

The process of using a median filter to remove salt-and-pepper noise involves defining a window, sliding the window over the image, computing the median of the pixel values within the window, and replacing the center pixel value with the median value. The algorithm shows following:

```
Data: Input image $I$; Specific size $m$
Result: Output image $J$
1 for Every segment $S = I(i : i + m, j : j + m)$ in $I$ do
2     Filter the segment by median filter;
3 end
4 Convert matrix to image;
```

The process of removing salt-and-pepper noise is similar to the process of local histogram equation. The only difference is utilizing numpy function median during processing the image segment.

```
output_median[y:y + seg_height, x:x + seg_width] = np.median(seg)
```

Alternatively, we also consider a different method of mean filter. The process for the mean filter is as following:

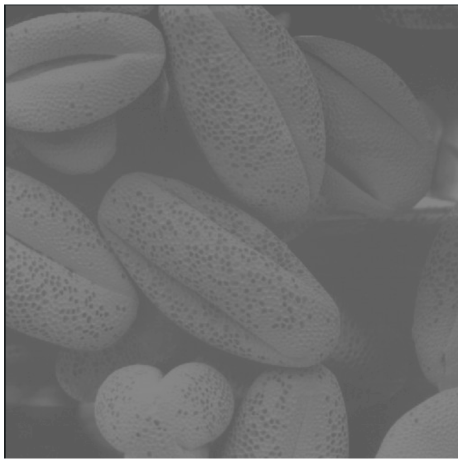
```
output_mean[y:y + seg_height, x:x + seg_width] = np.mean(seg)
```

3.2 Result and Analysis

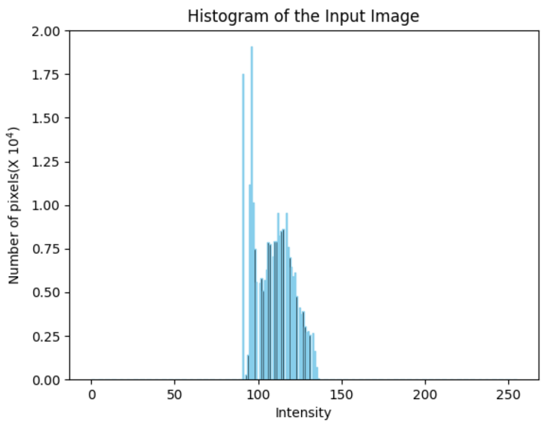
In the histogram equation, we process the same function in a dark image and a light image. Based on the experimental process described, it seems that the initial image had a low contrast, which was improved by histogram equalization as shown in Fig.1. Then, when the same process was applied to a darker version of the same image, a similar result was achieved with a moderate gray level. Furthermore, it appears that the histogram of the original image was biased towards either the left or right side, but after applying histogram equalization, the distribution became more uniform.

These observations can be explained by the nature of histogram equalization, which is a technique used to enhance the contrast of an image by redistributing its pixel intensities. When an image has low contrast, it means that its pixel intensities are clustered in a narrow range. Histogram equalization works by stretching this range to cover the full dynamic range of the image, thereby increasing its contrast. This can be seen in the improved contrast of the initial image after histogram equalization.

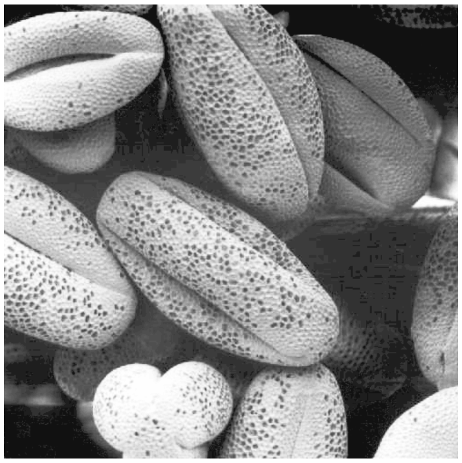
For the darker image, histogram equalization was used to stretch its pixel intensity range to cover the full dynamic range, resulting in a moderate gray level. The original image's histogram distribution bias could be due to lighting, compression, or inherent image characteristics. Histogram equalization can redistribute pixel intensities based on frequency to mitigate these biases and achieve a more uniform distribution.



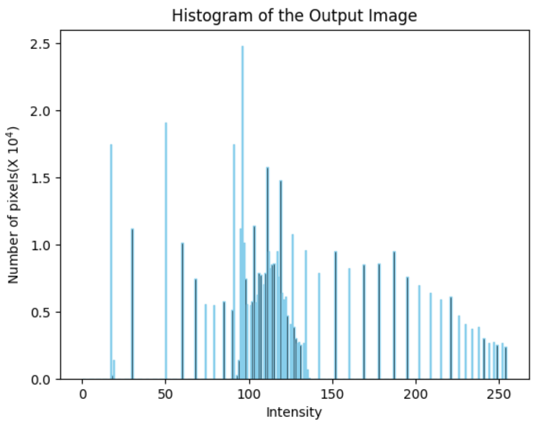
(a) Q3_1_1



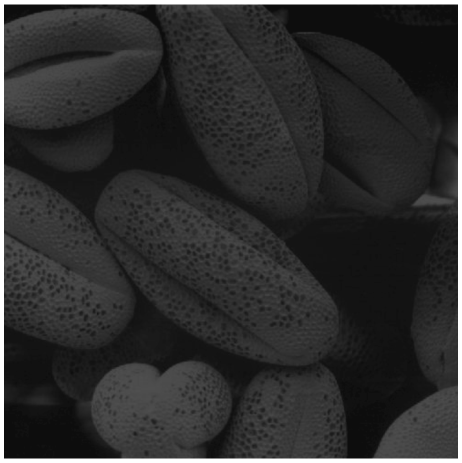
(b) Q3_1_1 histogram



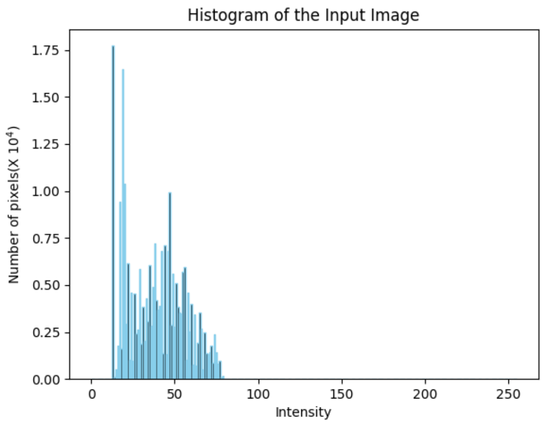
(c) Q3_1_1_out



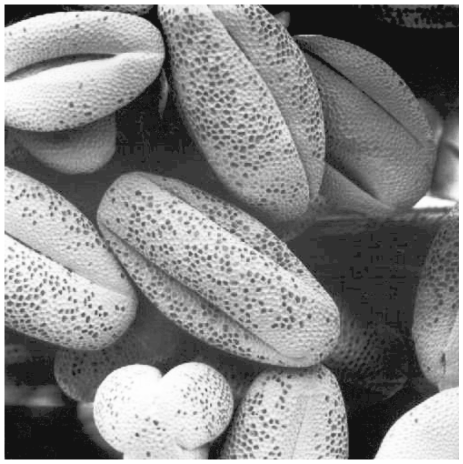
(d) Q3_1_1_out histogram

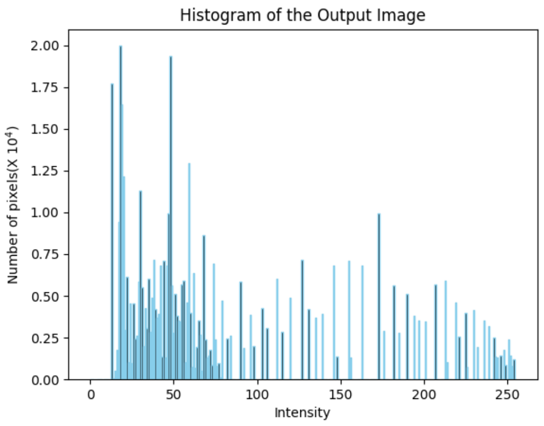


(e) Q3_1_2



(f) Q3_1_2 histogram


(g) Q3_1_2_out

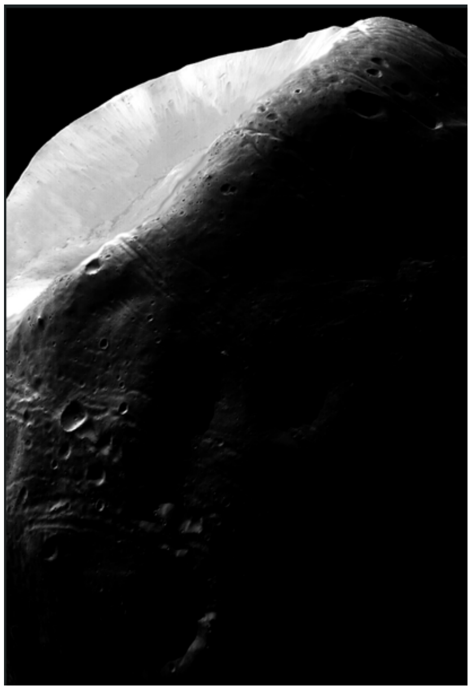


(h) Q3_1_2_out histogram

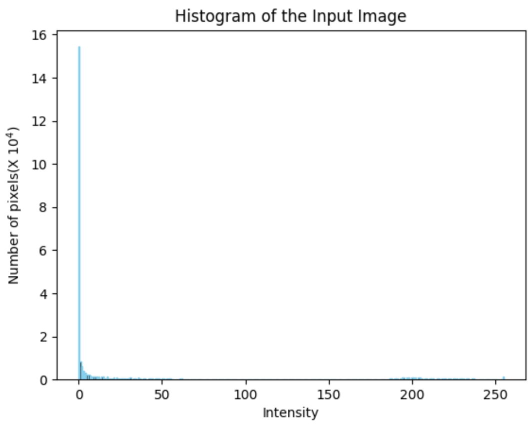
Figure 1: The figures of processed image and histogram

Regarding histogram matching, we generate a specific histogram for each image using the array_to_hist() function. In this lab, we have chosen two images - one with concentrated grayscale in the middle range, and the other with uniform grayscale. The results show that in the original image, the grayscale is mostly concentrated in the low area as shon in Fig.1. After processing with mask1, the grayscale is transferred to the middle area and the image shows more detail in the dark areas as shown in Fig.3. On the other hand, mask2 causes the grayscale of the input image to shift to the right, resulting in a lighter image with more detail in the dark areas as shown in Fig.4.

Histogram equalization can expand the dynamic range of an image and improve its contrast and sharpness, but it may not work well for images with scattered grayscale distribution. On the other hand, histogram matching can better control the brightness distribution of an image and preserve more image details, but it requires a suitable target histogram and may lead to image distortion if the target histogram is not properly selected.

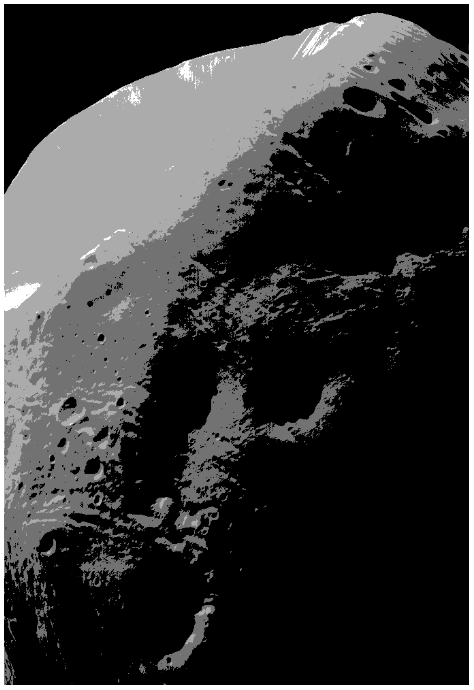


(a) Q3_2

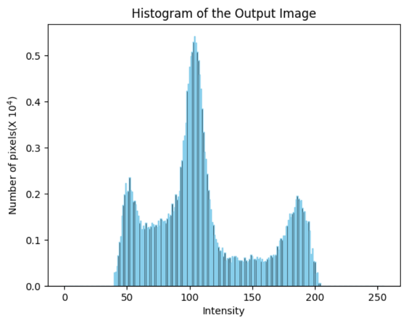


(b) Q3_2 histogram

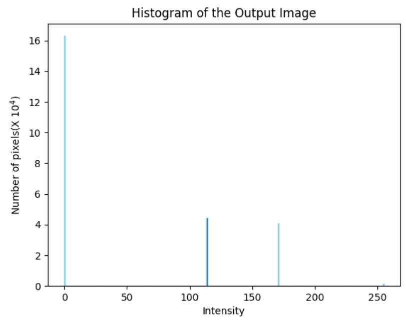
Figure 2: The figures of origin image and histogram



(a) Q3_2_out_mask1

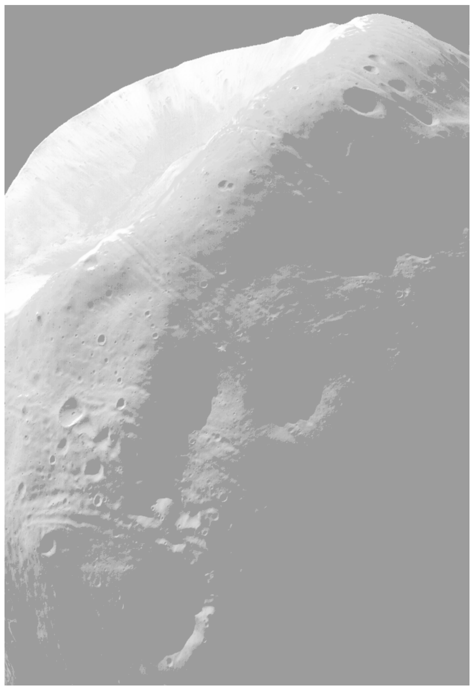


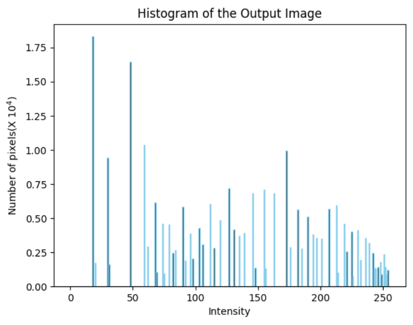
(b) mask1 histogram



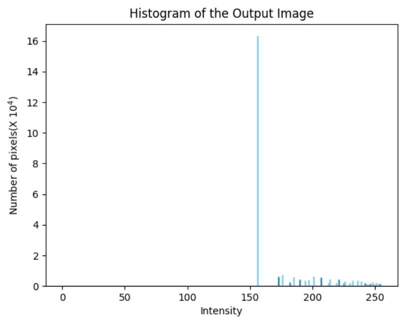
(c) Q3_2_out histogram

Figure 3: The figures of histogram matching image with mask1 and histogram


(a) Q3_2_out_mask2



(b) mask2 histogram



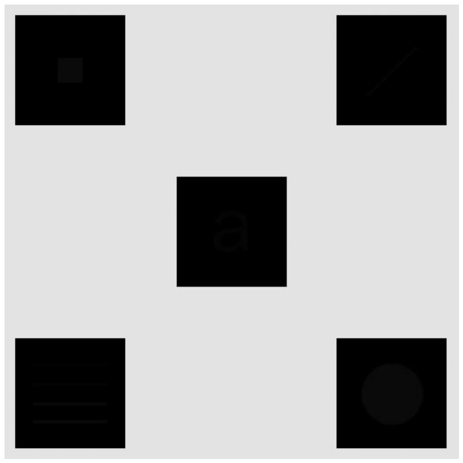
(c) Q3_2_out histogram

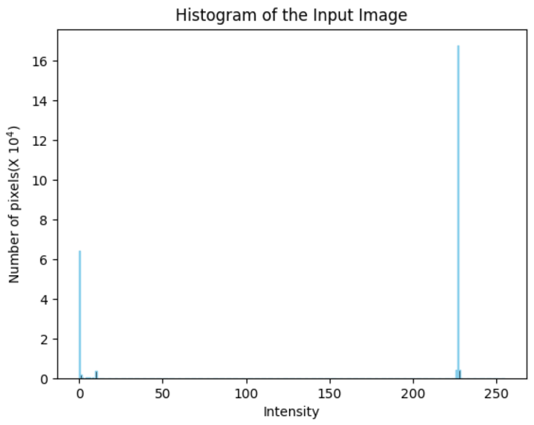
Figure 4: The figures of histogram matching image with mask2 and histogram

During my experiment of Q3, I processed an image using histogram equalization but found that it was not able to recover the features of dark blocks in the image. However, when I used a local 3x3 histogram equalization method, the image features were well restored and the overall image color was lighter as shown in Fig.5. The reason for this could be that the global histogram equalization method did not consider the local contrast of the image, and thus was not able to accurately enhance the details of the dark blocks. In contrast, the local histogram equalization method takes into account the local image information, which allows for a more precise adjustment of the contrast and brightness of the image. As a result, the local method was able to restore the image features and lighten the overall color without sacrificing image quality.

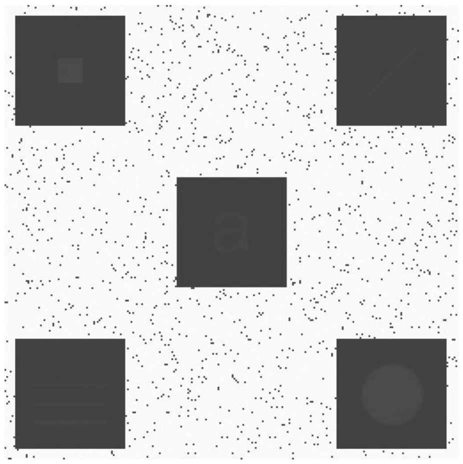
In my experiment on removing salt and pepper noise from images, I used two methods: a median filter which is introduced by tutorial and an averaging filter as an alternative method. I found that the averaging filter was unable to remove the salt and pepper noise, while the median filter was effective in removing it. Moreover, I observed that as the size of the filter window increased, the resulting image after filtering became increasingly blurred as shown in Fig.6.

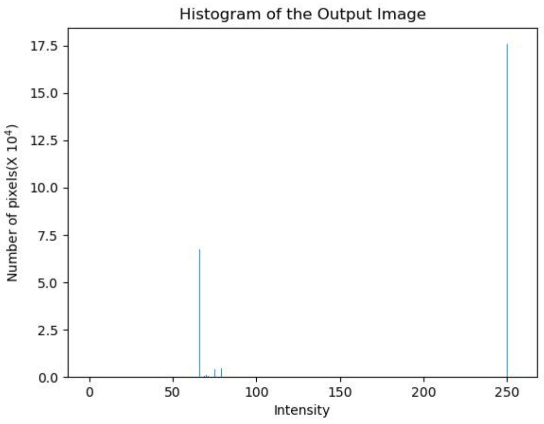
The reason for this phenomenon is that the averaging filter calculates the average pixel intensity values within a given window, which can result in a smoothing effect on the image, but cannot effectively remove the impulse noise caused by salt and pepper noise. In contrast, the median


(a) Q3_3

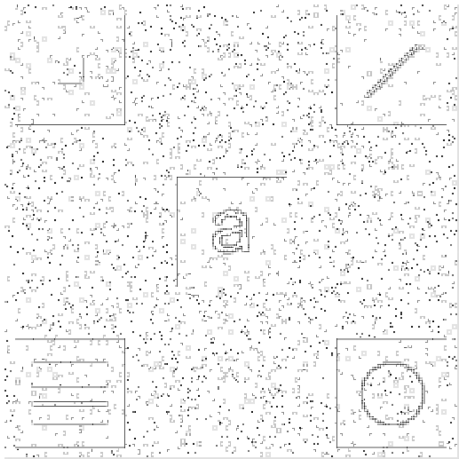


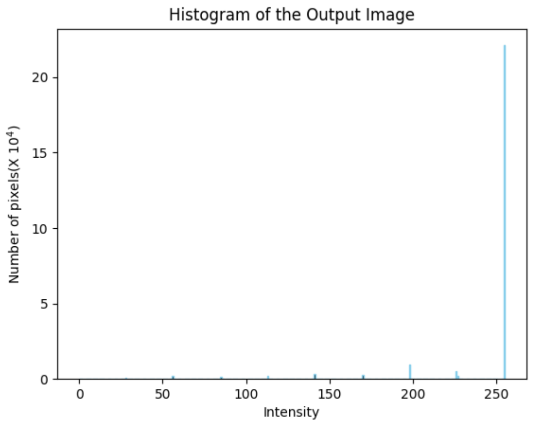
(b) Q3_3 histogram


(c) Q3_3_equ



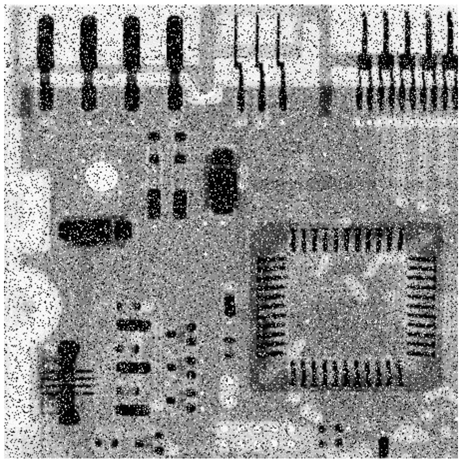
(d) Q3_3_equ histogram


(e) Q3_3_local

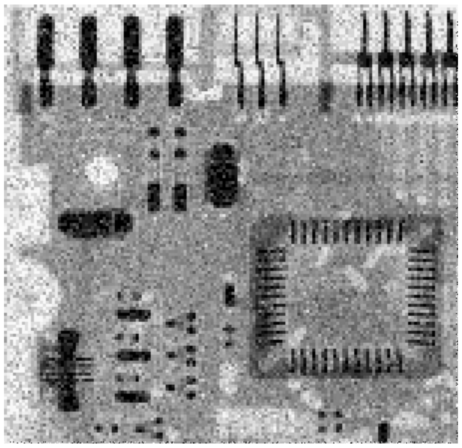


(f) Q3_3_local histogram

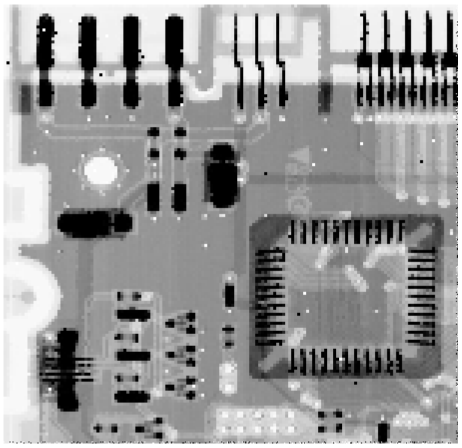
Figure 5: The figures of local histogram equation image and histogram



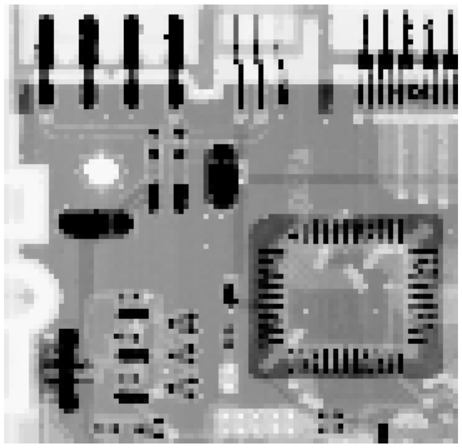
(a) Q3_4



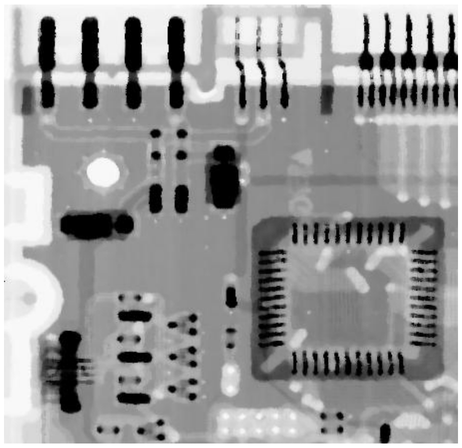
(b) Q3_4_mean: 3*3



(c) Q3_4_median: 3*3



(d) Q3_4_median: 5*5



(e) Q3_4_median: 10*10

Figure 6: The figures of SAP noise

filter replaces the pixel intensity value in a given window with the median value, which can effectively suppress impulse noise while preserving the image details.

However, as the window size increases, the median filter may also cause blurring in the image because the larger window size can cause the filter to lose the details of the image. Therefore, it is important to choose an appropriate window size for the median filter to balance the trade-off between noise reduction and image detail preservation.

3.3 Efficiency

After comparing the running times of histogram equalization, histogram matching, 3x3 local histogram equalization, and 3x3 median filtering algorithms, it was found that histogram equalization had the shortest time, followed by histogram matching, median filtering had slightly longer time than histogram matching, and local histogram equalization had the longest time as shown in Table.1. This finding is likely due to the fact that histogram equalization and histogram matching are both global image processing techniques that do not require iteration or processing of each pixel in the image. In contrast, median filtering and local histogram equalization require iteration over each pixel or a local region of pixels in the image, resulting in longer processing times. The size of the local region used in local histogram equalization may also affect the processing time, as a larger region requires more computation. I also noticed that the runtime of local histogram

Table 1: The time cost of all algorithm (unit:s)

Histogram Equation	Histogram Matching	Local Histogram Equation(3*3)	Median Filter(3*3)
0.156	0.286	0.656	0.303
0.161	0.285	0.636	0.309
0.161	0.296	0.631	0.306

equalization algorithm using a 5x5 window is less than that of using a 3x3 window as shown in Table.2. This phenomenon can be explained by the fact that a larger window size in local histogram equalization algorithm can result in a higher degree of smoothing and reduced local contrast, which may lead to fewer intensive operations and consequently reduce the overall runtime of the algorithm. Additionally, the smaller window size of 3x3 may require more iterations to cover the same image area compared to the 5x5 window, which could contribute to the observed difference in runtime. Besides, the execution time of 5x5 median filter is lower than that of 3x3

Table 2: The time cost of Local Histogram Equation with varying size(unit:s)

Local Histogram Equation(3*3)	Local Histogram Equation(5*5)
0.656	0.379
0.636	0.379
0.631	0.385

median filter, while the execution time of the average filter is lower than that of the median filter as shown in Table.3. The reason for this phenomenon is that the larger the window size of the filter, the more neighboring pixels need to be processed, which will increase the computational complexity and lead to longer execution time. Therefore, the 5x5 median filter, which has a larger window size than the 3x3 median filter, may take less time to process due to better noise reduction performance. On the other hand, the average filter is a simple linear filter that only needs to calculate the mean value of the neighboring pixels, which results in faster execution time compared to median filter that sorts the pixel values.

Table 3: The time cost of salt-and-pepper algorithms (unit:s)

Median Filter(3*3)	Averaging Filter(3*3)	Median Filter(5*5)
0.303	0.135	0.112
0.309	0.142	0.113
0.306	0.137	0.116

4 Conclusion

Based on the experiments conducted, we can conclude that each of the four image processing techniques, namely histogram equalization, histogram matching, local histogram equalization, and removal of salt-and-pepper noise, has its unique strengths and weaknesses in different scenarios.

Histogram equalization enhances image contrast by redistributing pixel intensity values, making use of the dynamic range for a more visually balanced result. However, it may also amplify noise, leading to unwanted artifacts.

Histogram matching matches the input image histogram with a reference image, transferring color or contrast characteristics to create a similar look and feel. It requires a reference image, which may not always be available or suitable.

Local histogram equalization applies the technique to small image regions to preserve local details and avoid over-amplification of noise. It's useful for images with non-uniform illumination or texture, but may be computationally expensive and produce visible artifacts at the edges of the regions.

Removing salt-and-pepper noise is crucial when dealing with affected images. Techniques like median filtering can be used, depending on the noise level, image size, and computational resources available. However, the removal of noise may affect the image quality and should be used with heed.

In conclusion, choosing the right image processing technique depends on the specific requirements of the task at hand. One should carefully consider the trade-offs between computational efficiency, visual quality, and noise characteristics. It is also important to have a good understanding of the characteristics of the input images, such as their size, content, and noise level, to choose the most suitable technique. Ultimately, image processing is both an art and a science, and one should always strive for a balance between creativity and technical rigor.