SUSTech_EE326_lab3

Histogram Processing

11911521钟新宇

1. Introduction

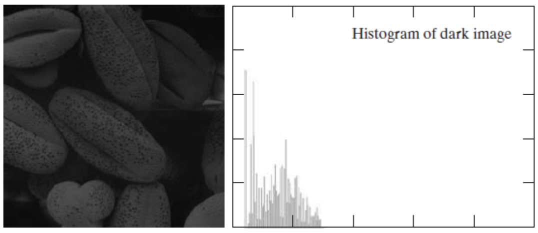
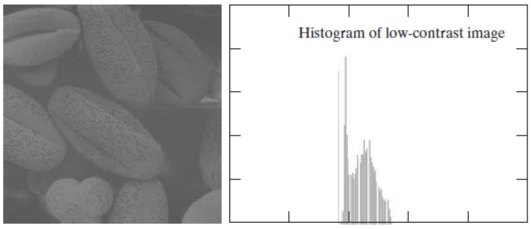
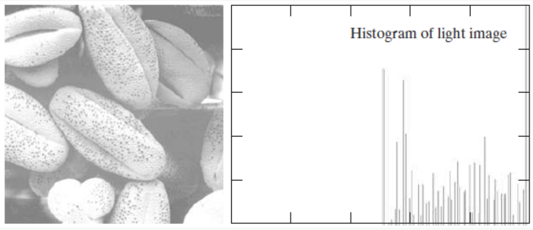
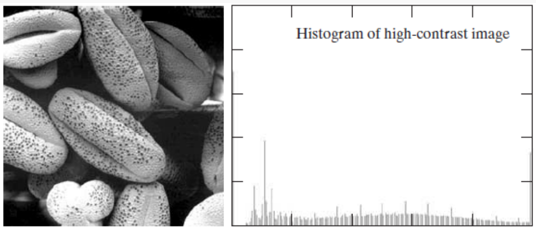
Histograms are the basis for numerous spatial domain processing techniques. In this lab, I will learn histogram processing algorithms such as histogram equalization, histogram matching, local histogram equalization, and noise removal, and then write python scripts to implement these functions.

2. Histogram Equalization

2.1 Requirements and Analysis

The histogram of a digital image with intensity levels in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$, where r_k is the kth intensity value and n_k is the number of pixels in the image with intensity r_k.

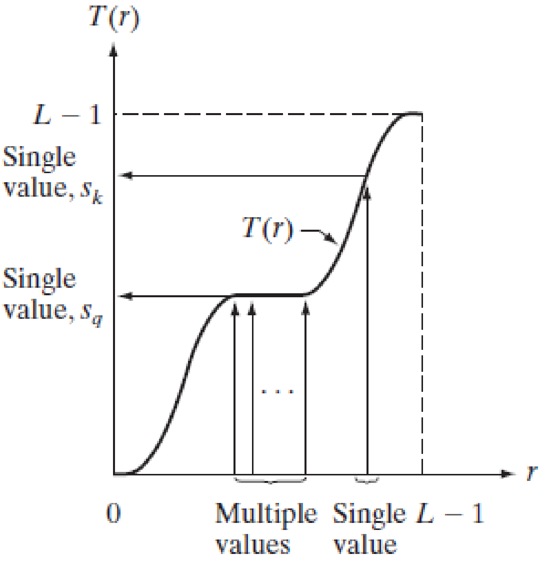
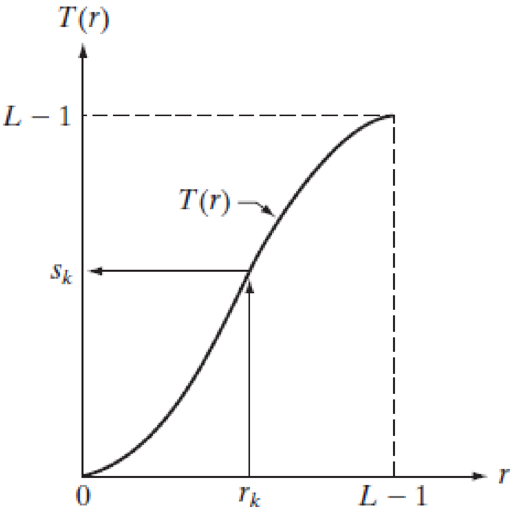
Ideally, the histogram of an image should be evenly distributed. In practice, however, we may find that some images are too dark or too bright overall, either due to camera malfunction or incorrect photographer handling. The histograms of these images will be concentrated in areas with low or high gray values. When we face such an image, in order to enhance the details in it, we can consider using the histogram equalization method.

Before equalization	After equalization
	
	

2.2 Algorithm Derivation

We assume that r represents the intensity order of the image and the intensity is up to $L-1$ and down to 0. Therefore, the distribution of r ranges from $[0, L-1]$. In order to achieve histogram equalization, we need to design an intensity mapping function, and consider denoting this transformation function by $s = T(r)$.

To ensure that the mapping transformations are one-to-one mappings, i.e., to avoid mapping inversions, we agree that $T(r)$ is a strictly monotonically increasing function, as follows.

Monotonically increasing function	Strictly monotonically increasing function
 $T(r)$ $L-1$ Single value, s_k $T(r)$ Single value, s_q . . . r 0 Multiple values Single value $L-1$	 $T(r)$ $L-1$ $T(r)$ s_k r 0 r_k $L-1$

The intensity levels in an image may be viewed as random variables in the interval $[0, L-1]$. A fundamental descriptor of a random variable is its probability density function (PDF). Let $p_r(r)$ and $p_s(s)$ denote the PDFs of r and s, respectively, where the subscripts on p are used to indicate that p_r and p_s are different functions in general. A fundamental result from basic probability theory is that if p_r and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed (mapped) variable s can be obtained using the simple formula.

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| \tag{13}$$

Since the transformation function of particular importance in image processing has the form

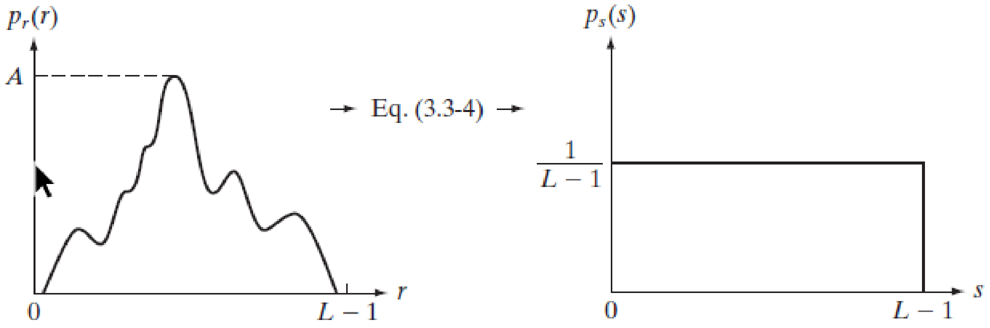
$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{14}$$

and the PDF of the output intensity variable, s, is determined by the PDF of the input intensities and the transformation function, we have

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = (L-1)\frac{d}{dr}\left[\int_0^r p_r(w)dw\right] = (L-1)p_r(r) \tag{15}$$

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| = p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right| = \frac{1}{L-1} \tag{16}$$

As shown in Equation 4, after the intensity mapping transformation, the output intensity histogram s shows a uniform distribution. This means that we achieve histogram equalization.



We can summarize the steps as follows.

- Calculate the histogram of the probability distribution
- Calculate the histogram of the cumulative distribution
- Use the histogram of the cumulative distribution to correct the intensity of the output image

2.3 Algorithms for discrete data

It is well known that computers can only handle discrete data, so we need to modify the above formula.

For discrete variables, we can describe its probability in this form

$$p_r(r_k) = \frac{n_k}{MN} \tag{17}$$

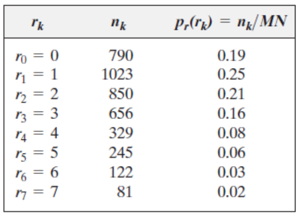
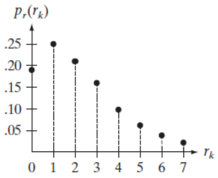
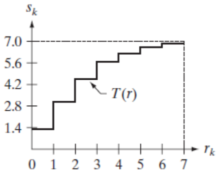
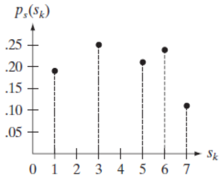
where MN is the total number of pixels in the image, n_k is the number of pixels that have intensity r_k.

The discrete form of the transformation in Eq. 2 is

$$\begin{aligned} s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \\ \frac{(L-1)}{} \sum^{k} \quad k = 0, 1, 2, \ldots, L-1 \end{aligned} \tag{18}$$

$$= \frac{\quad}{MN} \sum_{j=0} n_j \quad k = 0, 1, 2, \ldots, L-1$$

For example, given n_k , we can find s_k

<table>
<tr><th>n_k</th><th>$p_r(r_k)$</th><th>s_k</th><th>$p_s(s_k)$</th></tr>
<tr>
<td><table><tr><th>r_k</th><th>n_k</th><th>$p_r(r_k) = n_k/MN$</th></tr><tr><td>$r_0 = 0$</td><td>790</td><td>0.19</td></tr><tr><td>$r_1 = 1$</td><td>1023</td><td>0.25</td></tr><tr><td>$r_2 = 2$</td><td>850</td><td>0.21</td></tr><tr><td>$r_3 = 3$</td><td>656</td><td>0.16</td></tr><tr><td>$r_4 = 4$</td><td>329</td><td>0.08</td></tr><tr><td>$r_5 = 5$</td><td>245</td><td>0.06</td></tr><tr><td>$r_6 = 6$</td><td>122</td><td>0.03</td></tr><tr><td>$r_7 = 7$</td><td>81</td><td>0.02</td></tr></table></td>
<td> </td>
<td> </td>
<td> </td>
</tr>
</table>

2.4 Python Script

In this lab, we use the functions in the `opencv` library to read and write images.

```
img = np.array(cv2.imread(input_image, cv2.IMREAD_GRAYSCALE))
```

The `numpy` library provides the `np.histogram()` function, which can help us generate histograms.

```
numpy.histogram(a, bins=10, range=None, normed=None, weights=None, density=None)
    # a : array_like输入数据。直方图是在展平的数组上计算的。
    # bins : int 或 sequence of scalars 或 str, 可选
    #    如果 bins 是一个 int，则它定义给定范围内的等宽 bin 数（默认为 10 个）。
    #    如果 bin 是序列，则它定义 bin 边缘（包括最右边）的单调递增数组，从而允许非均匀的 bin 宽度。 1.11.0 版中的新功能。
    #    如果 bin 是字符串，则它定义用于计算最佳 bin 宽度的方法，如 histogram_bin_edges 所定义。
    # range : (float, float), 可选bin 的上下范围。如果未提供，则范围只是 (a.min(), a.max())。超出范围的值将被忽略。
    # normed : bool, 可选。
    # weights : array_like, 可选一组与 a 形状相同的 weights。每个中的每个值仅将其相关权重分配给仓位计数（而不是 1）。
    # density : bool,如果为 False，则结果将包含每个 bin 中的样本数。如果为 True，则结果为 bin 处的概率密度函数的值，将其归一化以使该范围内的积分为 1。

```

We specify that there are 256 intervals, meaning that `bins=257`, and we note that `density=True` returns the value of the probability density function.

```
input_hist, _ = np.histogram(img.flat, bins=bins, density=True)
output_hist, _ = np.histogram(output_image.flat, bins=bins, density=True)
```

For the mapping transformation function described in Equation 2, we can use accumulation instead of integration.

```
s = np.array(np.zeros(256))
for i in range(L):
    s[i] = (L - 1) * sum(input_hist[:i + 1])
```

After obtaining the equalized intensity array s, we only need to look up the gray values of the original image in it to obtain the gray values of the corresponding positions of the output image.

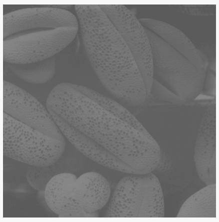
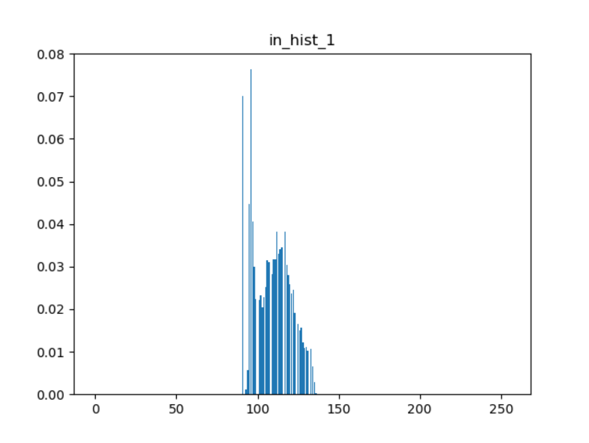
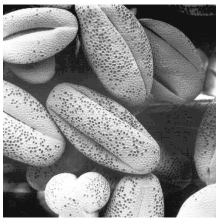
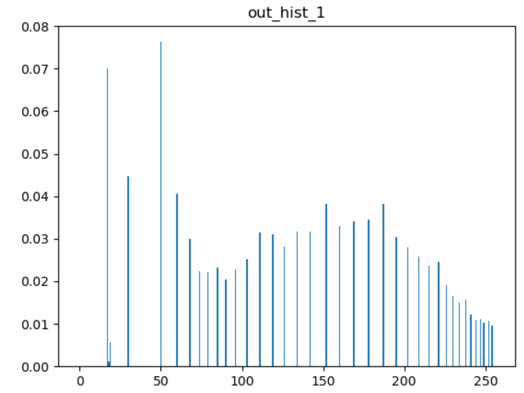
```
output_image = np.array(np.zeros((m, n)))
for i in range(m):
    for j in range(n):
        output_image[i][j] = s[img[i][j]]
```

All codes are as follows.

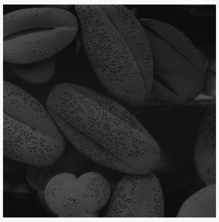
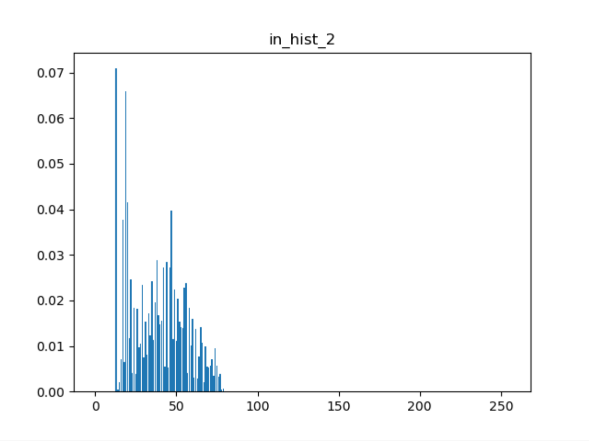
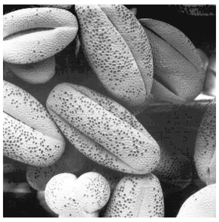
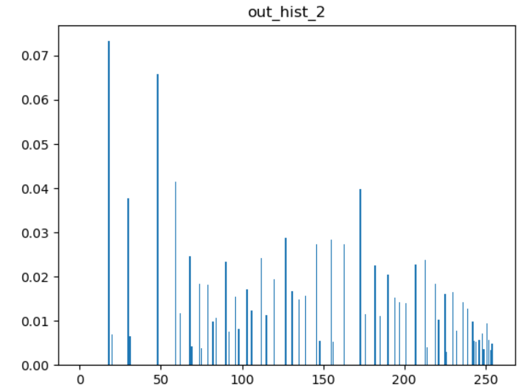
```
def hist_equ_11911521(input_image):
    img = np.array(cv2.imread(input_image, cv2.IMREAD_GRAYSCALE))
    m, n = img.shape
    L = 256
    bins = range(L + 1)
    input_hist, _ = np.histogram(img.flat, bins=bins, density=True)

    # s = np.array([(L - 1) * sum(input_hist[:k + 1]) for k in range(L)])
    s = np.array(np.zeros(256))
    for i in range(L):
        s[i] = (L - 1) * sum(input_hist[:i + 1])

    # output_image = np.array([s[r] for r in img], int).reshape(img.shape)
    output_image = np.array(np.zeros((m, n)))
    for i in range(m):
        for j in range(n):
            output_image[i][j] = s[img[i][j]]

    output_hist, _ = np.histogram(output_image.flat, bins=bins, density=True)

    return output_image, output_hist, input_hist

```

2.5 Results

- Q3_1_1.tif

Before equalization		
After equalization		

- Q3_1_2.tif

Before equalization		
After equalization		

The results of histogram equalization are shown above. We can observe that the overly bright and dark images are equalized to enhance the less distributed intensities in the original image and suppress the more distributed intensities in the original image. The probability density distribution of the intensities of the processed images is more uniform.

3. Histogram Match

3.1 Requirements and 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. When automatic enhancement is desired, this is a good approach because the results from this technique are predictable and the method is simple to implement.We show in this section that 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.

3.2 Algorithm Derivation

The principle of histogram matching is similar to that of histogram equalization. First, a random variable s is defined, and s is calculated based on the intensity probability density of the image

$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{19}$$

Then define a random variable z, and compute G(z) based on the known intensity probability density of the histogram

$$G(z) = (L-1)\int_0^z p_z(t)dt = s \tag{20}$$

Using condition $G(z) = T(r)$, we find the inverse function of s to derive the random variable z, which is the intensity level of the output image.

$$z = G^{-1}[T(r)] = G^{-1}(s) \tag{21}$$

In summary, we can summarize the histogram matching as the following steps.

- Calculate the histogram of the probability distribution of the input image
- Calculate the histogram of the cumulative probability distribution of the input image
- Calculate the histogram of the cumulative probability distribution of a particular histogram
- Calculate the histogram of the probability distribution of the output image
- Calculating the output image
- Calculate the CPDF of the histogram of the output image

3.3 Algorithms for discrete data

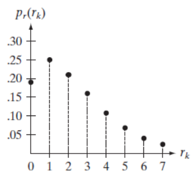
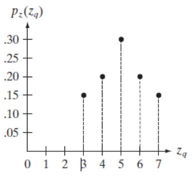
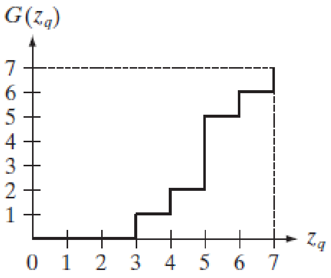
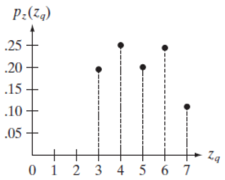
Similar to the algorithm for histogram equalization, we replace the integral in the above equation with a cumulative one. The results are shown below.

$$\begin{aligned} s_k = T(r_k) &= (L-1)\sum_{j=0}^{k} p_r(r_j) \\ &= \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \quad k = 0, 1, 2, \ldots, L-1 \end{aligned} \tag{22}$$

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i) \tag{23}$$

$$z_q = G^{-1}(s_k) \tag{24}$$

However, we do not have to perform the operation of the inverse function. Since the purpose of the inverse function is to find the nearest $G(z_k)$ to s_k, we can subtract each item in the array G from s_k separately and return the k with the smallest absolute value. And z(k) is the value of k.

Histogram of a image.	**Specified histogram.**	**Transformation function obtained from the specified histogram.**	**Result of performing histogram specification.**
			

3.4 Python Script

First, read the image data

```python
img = np.array(cv.imread(input_image, cv.IMREAD_GRAYSCALE))
m, n = img.shape
L = 256
bins = range(L + 1)
```

Calculate the histogram of the probability distribution of the input image

```python
input_hist, _ = np.histogram(img.flat, bins=bins, density=True)
```

Calculate the histogram of the cumulative probability distribution of the input image

```
s = np.array(np.zeros(L))
for i in range(L):
    s[i] = np.round((L - 1) * sum(input_hist[:i + 1]))
```

Calculate the histogram of the cumulative probability distribution of a particular histogram

```
gz = np.array(np.zeros(L))
for i in range(L):
    gz[i] = np.round((L - 1) * sum(spec_hist[:i + 1]))
```

Calculate the histogram of the probability distribution of the output image

```
 z = np.array(np.zeros(L))
 for i in range(L):
    z[i] = np.abs(gz - s[i]).argmin()
```

Calculating the output image

```
 output_image = np.array(np.zeros((m, n)))
 for i in range(m):
    for j in range(n):
        output_image[i][j] = z[img[i][j]]
```

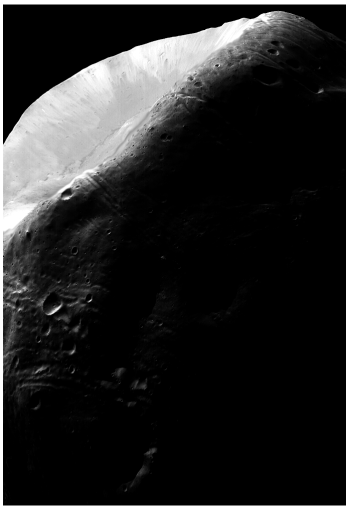
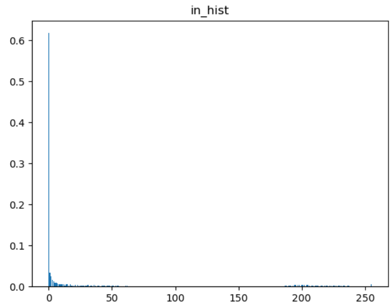
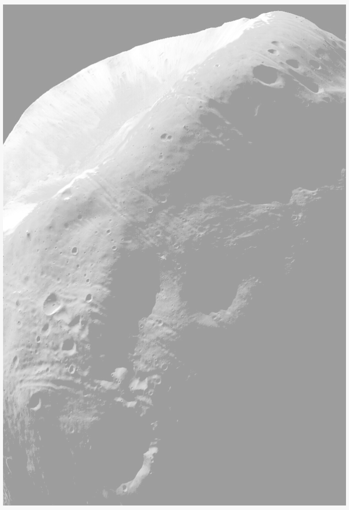
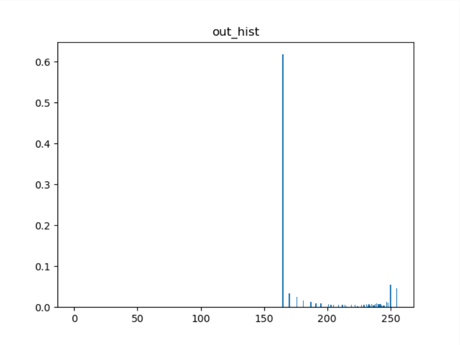
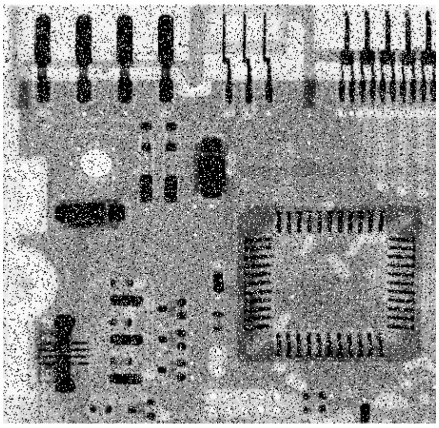
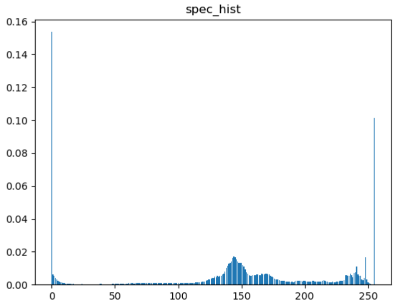
Calculate the CPDF of the histogram of the output image

```
output_hist, _ = np.histogram(output_image.flat, bins=bins, density=True)
```

All codes

```
def hist_match_11911521(input_image, spec_hist):
    img = np.array(cv.imread(input_image, cv.IMREAD_GRAYSCALE))
    m, n = img.shape
    L = 256
    bins = range(L + 1)

    input_hist, _ = np.histogram(img.flat, bins=bins, density=True)

    s = np.array(np.zeros(L))
    for i in range(L):
        s[i] = np.round((L - 1) * sum(input_hist[:i + 1]))

    gz = np.array(np.zeros(L))
    for i in range(L):
        gz[i] = np.round((L - 1) * sum(spec_hist[:i + 1]))

    # z = np.array([np.abs(gz - e).argmin() for e in s], int)
    # print(z)
    z = np.array(np.zeros(L))
    for i in range(L):
        z[i] = np.abs(gz - s[i]).argmin()

    output_image = np.array(np.zeros((m, n)))
    for i in range(m):
        for j in range(n):
            output_image[i][j] = z[img[i][j]]

    output_hist, _ = np.histogram(output_image.flat, bins=bins, density=True)

    return output_image, output_hist, input_hist
```

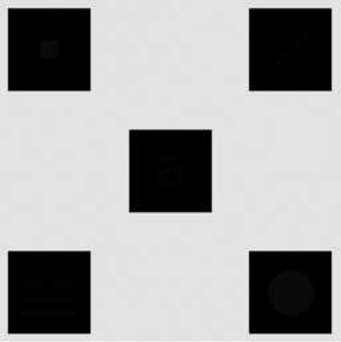
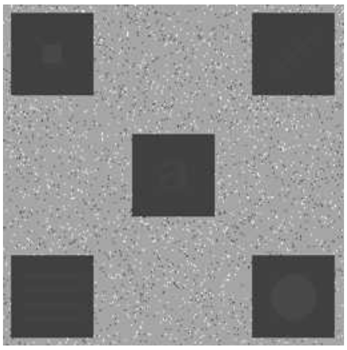
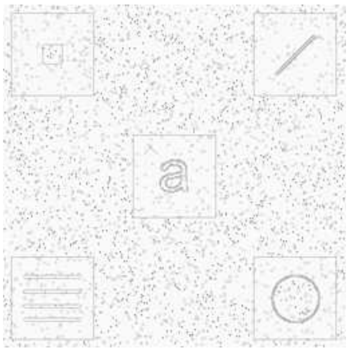
3.5 Results

	Image	Histogram
Before matching		
After matching		
spec_hist		

4. Local Histogram Equalization

4.1 Requirements and Analysis

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.

Original image	Result of global histogram equalization	Result of local histogram equalization
		

4.2 Algorithm Derivation

The local histogram equalization algorithm is also very simple. First, a neighborhood is determined, the histogram s of the cumulative distribution within the neighborhood is calculated, and the intensity of the neighborhood centroid is corrected using s. Then move to the next neighborhood.

We can summarize the steps as follows.

- Determine a neighborhood
- Calculate the histogram of the probability distribution within the neighborhood
- Calculate the histogram of the cumulative distribution within the neighborhood
- Use the histogram of the cumulative distribution to correct the intensity of the centroids
- Move to the next neighborhood

4.3 Python Script

Determine a neighborhood

```
local = np.zeros((m_size, m_size))
for k in range(i - local_len, i + local_len):
    for l in range(j - local_len, j + local_len):
        if 0 <= k < m and 0 <= l < n:
            local[k - (i - local_len), l - (j - local_len)] = img[k, l]
```

Calculate the histogram of the probability distribution within the neighborhood

```
local_hist, _ = np.histogram(local.flat, bins=bins, density=True)
```

Calculate the histogram of the cumulative distribution within the neighborhood

```
for k in range(L):
    s[k] = (L - 1) * sum(local_hist[:k + 1])
```

Use the histogram of the cumulative distribution to correct the intensity of the centroids

```
output_image[i][j] = s[img[i][j]]
```

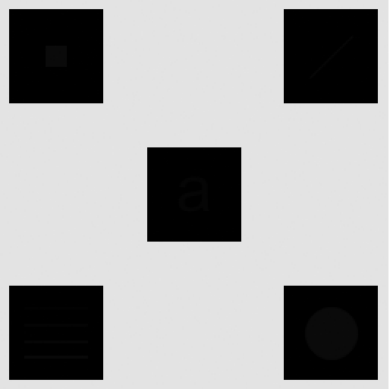
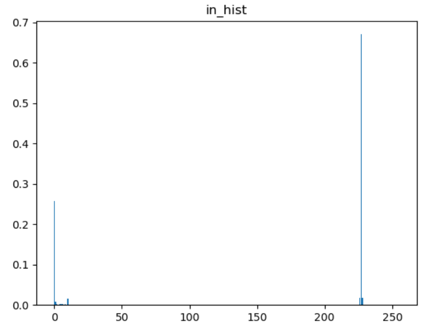
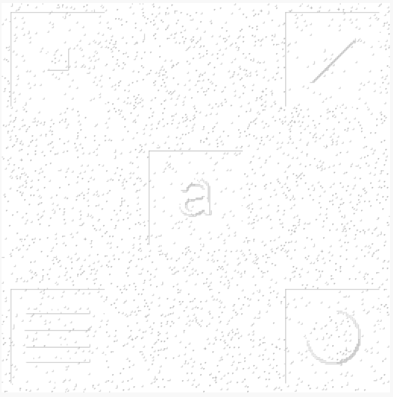
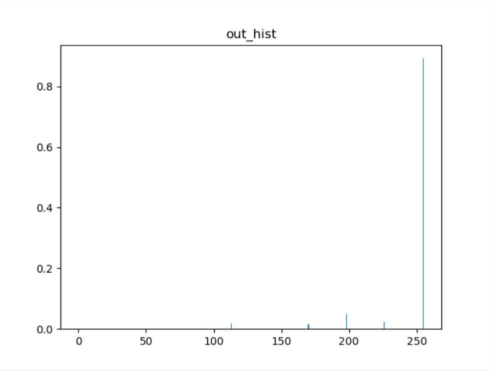
Move to the next neighborhood

```
for i in range(m):
    for j in range(n):
        ...
```

All codes are as follows.

```
def local_hist_equ_11911521(input_image, m_size=None):
    int_image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    img = np.array(int_image)
    m, n = int_image.shape
    L = 256
    local_len = int(m_size / 2)
    bins = range(L + 1)

    input_hist, _ = np.histogram(img.flat, bins=bins, density=True)

    s = np.array(np.zeros(L))
    output_image = np.array(np.zeros((m, n)))

    for i in range(m):
        for j in range(n):
            print("%d,%d" % (i, j))
            local = np.zeros((m_size, m_size))
            for k in range(i - local_len, i + local_len):
                for l in range(j - local_len, j + local_len):
                    if 0 <= k < m and 0 <= l < n:
                        local[k - (i - local_len), l - (j - local_len)] = img[k, l]

            local_hist, _ = np.histogram(local.flat, bins=bins, density=True)

            for k in range(L):
                s[k] = (L - 1) * sum(local_hist[:k + 1])

            output_image[i][j] = s[img[i][j]]

    output_hist, _ = np.histogram(output_image.flat, bins=bins, density=True)

    return output_image, output_hist, input_hist
```

4.4 Results

	Image	Histogram
Before matching		
After matching		

The results of local histogram equalization are shown in the figure above. During the running, we found that the script was very time consuming and took 756 seconds. This is because my algorithm uses a four-fold for loop. And the time complexity is $O(3 \cdot mn \cdot L)$.

```python
for i in range(m):
    for j in range(n):
        for k in range(i - local_len, i + local_len):
            for l in range(j - local_len, j + local_len):
        for k in range(L):
                ...
```

Therefore my algorithm needs to be improved by reducing the time complexity to speed up the process. However, I have not found a way to reduce the complexity effectively yet.

5. Reduce SAP Noise

5.1 Requirements and Analysis

Pepper noise is also called impulse noise. It is a black and white light and dark dot noise generated by the image sensor, transmission channel, decoding process, etc.. The black noise dots are figuratively called pepper noise, while the white noise dots are called salt noise. Generally these 2 kinds of noise appear at the same time, presented in the image is black and white miscellaneous dots. Pepper noise is often caused by image cutting, the most common algorithm to remove impulse interference and pepper noise is median filtering.

5.2 Algorithm Derivation

A more effective way to filter out pretzel noise is to apply median filtering to the signal. After removing the pretzel noise, a smoother signal can be obtained, and its effect is better than the mean filter, but of course, median filtering can also cause blurred edges and less sharp signals.

The principle of median filtering is: to select a part of the region pixel gray value to obtain its median value, and use the obtained median value to replace the pixel gray value in the region, so as to achieve the effect of smooth filtering. Since we require the median value, we need to be careful to choose an odd size range for the median calculation when selecting the region range.

The algorithm for calling the median filter to remove noise is similar to the local histogram equalization algorithm. They both create subregions in the original image. The local histogram equalization algorithm does histogram equalization for the pixel points in the neighborhood and updates the center point gray value. The median filtering denoising algorithm does median filtering on the pixel points in the sub-region and updates the centroid gray value.

5.3 Python Script

First, read the image data

```python
image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
```

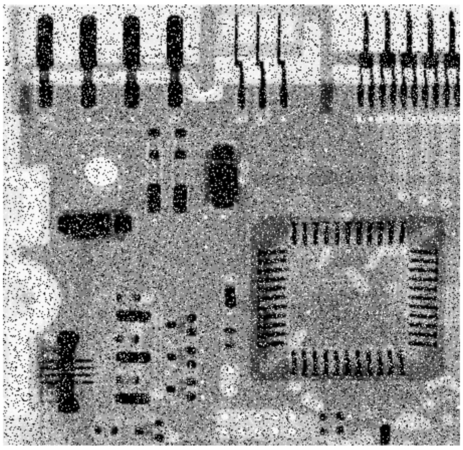
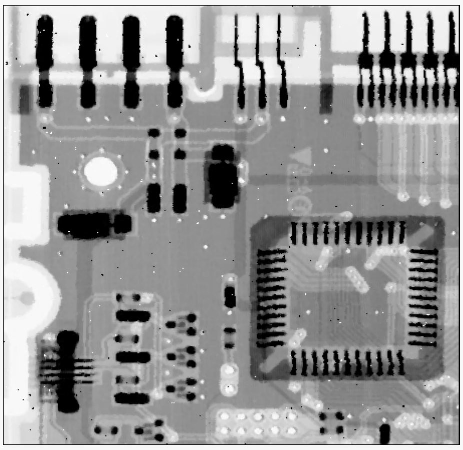
Call `np.median()` to perform median filtering

```python
for i in range(local_len, m - local_len):
    for j in range(local_len, n - local_len):
        output_image[i][j] = np.median(img[i - local_len:i + local_len + 1, j - local_len:j + local_len + 1])

```

All codes are as follows

```python
def reduce_SAP_11911521(input_image, n_size):
    image = cv2.imread(input_image, cv2.IMREAD_GRAYSCALE)
    img = np.array(image)
    m, n = img.shape
    local_len = int((n_size - 1) / 2)

    output_image = np.array(np.zeros((m, n)))

    if m - 1 - local_len <= local_len or n - 1 - local_len <= local_len:
        print("The parameter k is to large.")

    for i in range(local_len, m - local_len):
        for j in range(local_len, n - local_len):
            output_image[i][j] = np.median(img[i - local_len:i + local_len + 1, j - local_len:j + local_len + 1])

    return output_image
```

5.4 Results

	Image
Before matching	
After matching	

The result of removing the SAP noise is shown in the figure above. My algorithm removes the noise very well.

6. Summary

In this lab, I learned three histogram processing algorithms (e.g., histogram equalization, histogram matching, and local histogram equalization) and a noise reduction algorithm (median filtering), and wrote python scripts to implement these functions. In the process of experimenting, I understood the principles of the algorithms in depth. Of course, there are some problems with the scripts I wrote, which I will solve in future.