

## 1. Introduction

Vision is a major source of information for human beings. Earlier it was impossible to achieve but due to the development of new technologies it has been made possible. Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It also has wide applications in machine learning, signal processing, medical field and so on.

Although I am also studying the field of image recognition in my university, I usually use libraries to preprocess images. This is my first time to code the functions without libraries, which is very interesting. I hope this assignment will help me to understand the details of image processing algorithms. Learn more about it and use it in my research in the future.

## 2. Methods

- Histogram equalization

A histogram represents the frequency distribution of data. Histogram equalization is a technique to enhance a digital image in computer vision applications. It is especially effective in improving the visual quality of grayscale images. To have a good contrast, the following histogram characteristics are desirable:

- (1) the pixel intensities are uniformly distributed across the full range of values (each intensity value is equally probable).
- (2) the cumulative histogram is increasing linearly across the full intensity range.

Histogram equalization modifies the distribution of pixel intensities to achieve these characteristics.

- Median filter vs Adaptive median filter

### A. Median filter

This filter is ideal for eliminating unipolar or bipolar impulsive random noise, as is, in the latter case, the case of the noise called “salt and pepper”. On the other hand, image contrast is lost due to the filter’s own work, and in case of not choosing a good value for the kernel window size, this effect will be exacerbated, or else it will not remove the noise at all. How does it work? The median filter replaces each pixel with the median of the intensity levels of its neighbors.

### B. Adaptive median filter

This filter dynamically changes the window size of the filter during the filtering process according to preset conditions. Further, according to the conditions, it determines whether the current pixel is noisy,

thereby determining whether to replace the current pixel with the neighborhood median value.

It can deal with impulsive noise with high probability, smooth non-impulsive noise, protect image details as much as possible, and avoid image edge distortion.

- Color corrections

Color correction techniques are often used when an image is underexposed, overexposed, or color imbalance problems which can be result from the devices or the other conditions. It fixes color issues and makes footage appear as naturalistic as possible. The idea is for colors to look clean and real, as human eyes would see them in the real world. Color correction includes many approach to achieve, there are mainly two approach I would like to use to adjust image color:

- A. Mean and median adjustment

Adjust the image via the mean, median, maximum values per channel.

- B. Percentile adjustment

Filter the image based on specific percentile ranks per channel.

- Morphological image processing

Morphology is a comprehensive set of binary image processing operations that process images based on shapes. Morphological operations apply a structuring element (SE) to an input image, creating an output image of the same size.

- A. Erosion:  $A \ominus B$

It removes the boundaries of the foreground object in the image. It reduces the size of the object in the image.

- B. Dilation:  $A \oplus B$

It expands the boundaries of the foreground object which in turn increases the size of the object in the image.

- C. Opening:  $A \circ B = (A \ominus B) \oplus B$

The erosion operation first removes all the small noisy areas from the image and then dilation is applied to restore the original size of the object.

- D. Closing:  $A \cdot B = (A \oplus B) \ominus B$

This is useful for joining or filling small spots that are present in the foreground object, while retaining the size of the object.

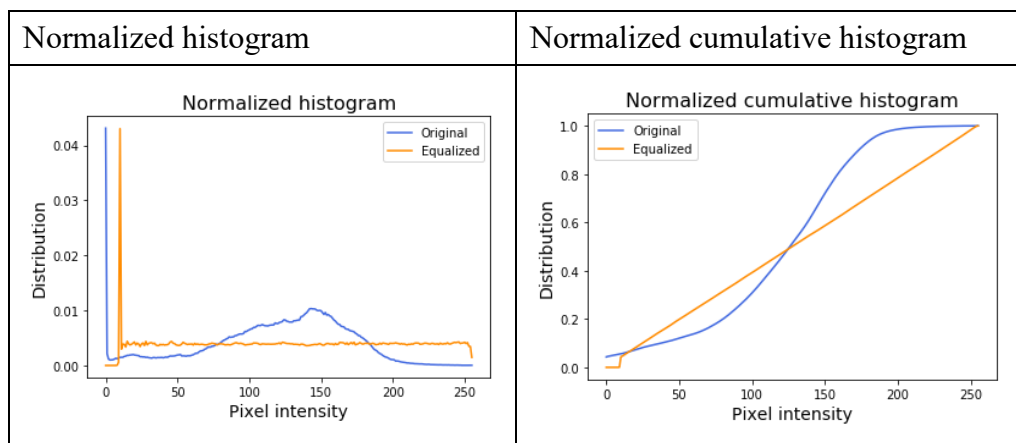
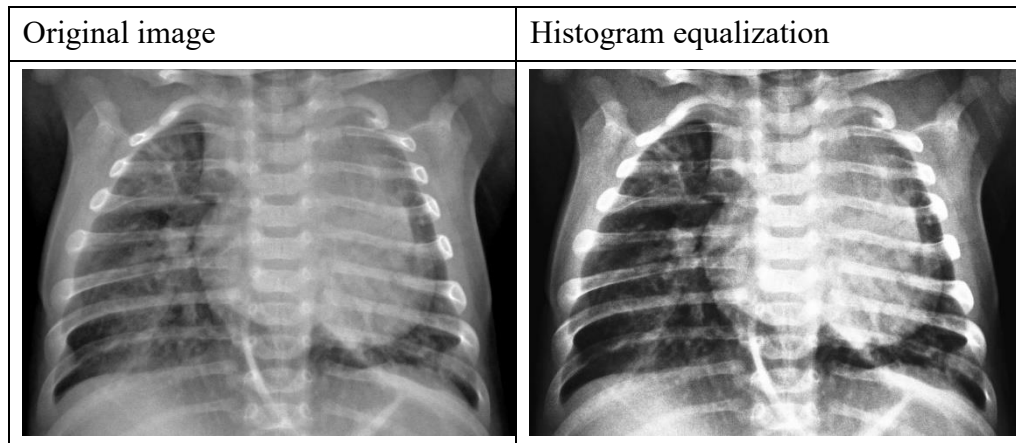
- E. Gradient:  $(A \oplus B) - (A \ominus B)$

The result of the gradient operation is the outline of the foreground object in the image.

### 3. Experiment results

- Histogram equalization

Histogram equalization has been applied extensively in medical imaging to improve the contrast of X-ray and MRI images. Such improvement enables more accurate medical diagnosis. To improve the contrast in a medical image, I have chosen a chest x-ray image as an example for my presentation.



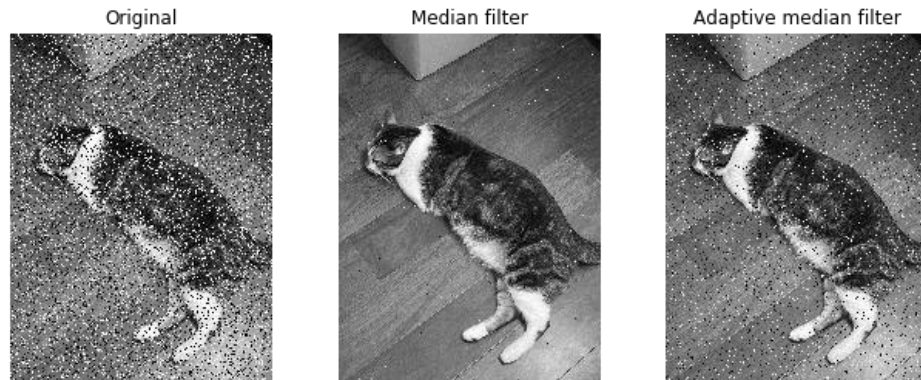
For the above results, the upper two images show the original and equalized images, improvement in contrast is clearly observed. The lower two images show the normalized histogram and normalized cumulative histogram, comparing original and equalized images. After histogram equalization, the pixel intensities are distributed across the whole intensity range better. The cumulative histograms are increasing linearly as expected, while exhibiting staircase pattern. This is expected as the pixel intensities of the original image were stretched into a wider range.

- Median filter vs Adaptive median filter

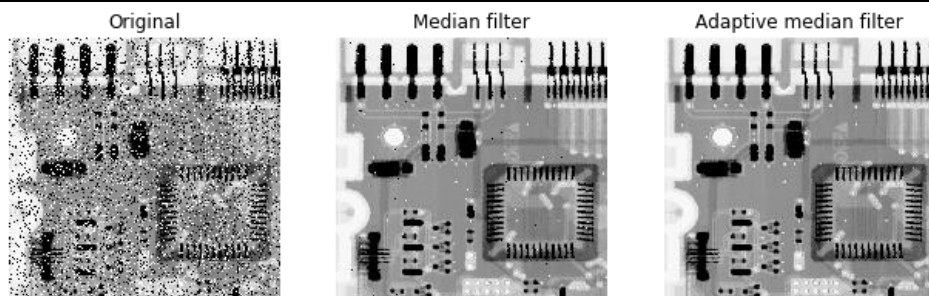
In this part, I have chosen a cat picture I took at my friend's house, but the image quality is relatively high, I added salt-and-pepper noise to my image for the purpose of testing. After I reviewed the course content, I found that the median filter is highly effective in removing salt-and-pepper noise. Therefore, I want to make a comparison between median filter and its

improved filtering technique, as known adaptive median filter.

My own image (cat)



Sample image of textbook



This result is a little bit tricky. Normally, the performance of the adaptive median filter should be better than that of the median filter. But from the test results of the images I took myself, it was not the case. It can be seen that the result image of median filter has less noise is preserved than the adaptive median filter. Later, I also used the sample image of textbook to test my code and it seemed to work fine. This problem and my explanation will be discussed in the next part.

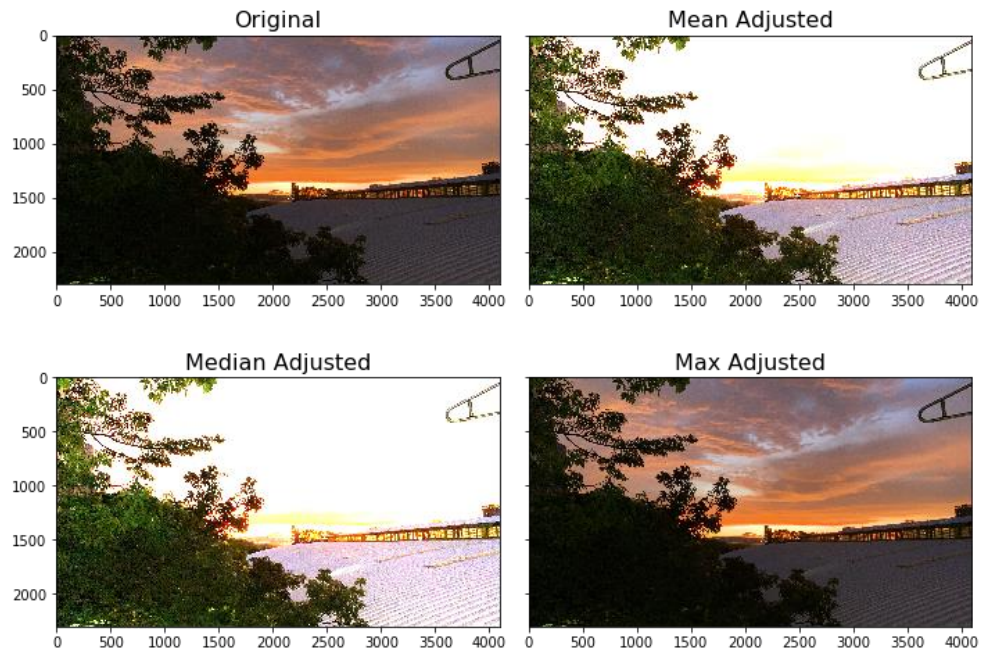
- Color corrections

For this image processing method, I used a darker image that was taken outside my lab at my university. From a purely visual perspective, we can see that the image has a red or orange color sky. To have a better understanding of the distribution of colors, I checked the statistics per channel.

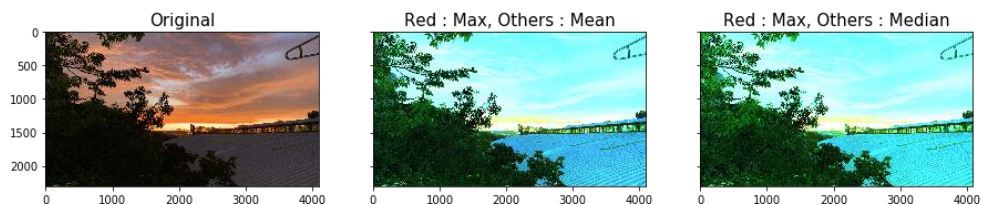
|       | Max | Mean      | Median | P_90  | P_95  | P_99  |
|-------|-----|-----------|--------|-------|-------|-------|
| Red   | 255 | 95.800207 | 69.0   | 202.0 | 222.0 | 253.0 |
| Green | 255 | 72.809488 | 58.0   | 139.0 | 152.0 | 204.0 |
| Blue  | 253 | 61.505131 | 59.0   | 128.0 | 150.0 | 188.0 |

I observed that the highest value of the channel is red. We can see that there is indeed a red overcast on this image. I want to turn a red overcast into

a clear blue sky. Although this will probably not work, I still want to try to adjust this image by the mean, median, maximum values per channel.



It can be seen from the above four images that the problem is not resolved by using this method. Maybe the image is clearly much brighter, the image appears to be slightly overexposed. One possible way to tackle this issue is by focusing on the red channel. Then, I tried to adjust the red channel using its max value, and all the other channels are adjusted using their mean and median values.



It seems that we are on the right track. The image has clearly been rid of the red overcast. However, I see a much more pronounced overcast of blue, it still doesn't look very natural. This implies that I have adjusted red far too much, or maybe insufficiently adjusted green and blue. I would filter the image based on specific percentile ranks per channel.





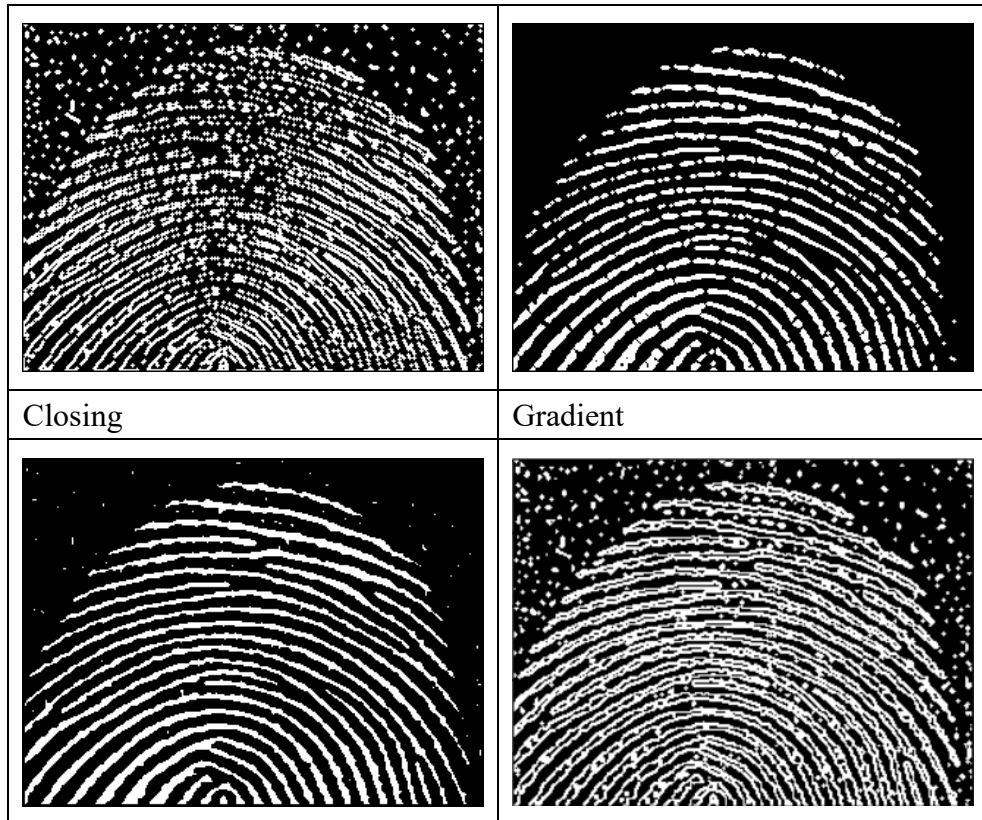


The improvement in image enhancement is very noticeable. Not only is the red shadow greatly reduced, but also the overexposure of the other channels is kept to a relatively low level.

- Morphological image processing

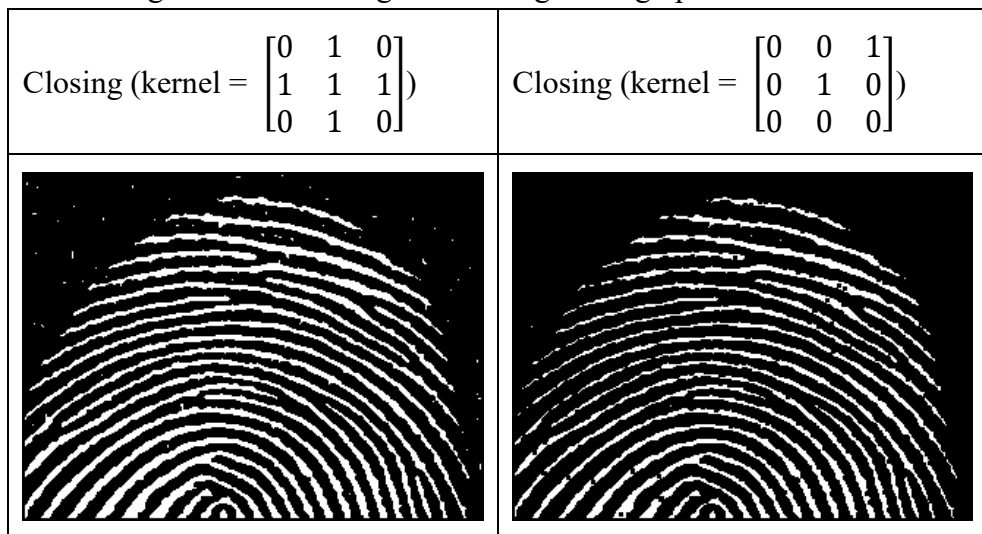
In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. Because I think this technology is very important in the field of fingerprint recognition, I also use a sample image of the textbook as my dataset. The two most widely used operations are Erosion and Dilation, but in this assignment, I would like to challenge some combinations of them and modify the kernel matrix to compare.

|   |  |
|---|--|
| Original image  | Erosion  |
|  |  |
| Dilation  | Opening  |



In the above experiments, the operational kernel sizes are 3, and the kernels of dilation operation are  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ . At first glance, the results of all

operations are some defects, for example, the fingerprint after using opening operation has too much gaps in the image, and there is also a little bit of noise in the background of the image after using closing operation.



For the closing operation, I tried to modify the kernel matrix to improve its performance, but it seem to be not working very well. The noise in the background were successfully removed, but the sharpness of the fingerprint

decreased.

#### 4. Discussion

In the first part, I would explain and give my thoughts according to different processing methods. In the second section, I would list a few problems that I think it can be improved by other methods, and of course there are also one or two problems that I have no way to solve.

- Observations / Interpretations

- Histogram equalization

I believe that this is a basic method to improve the image quality, but I think it is also important to implement without OpenCV, I can clearly understand its algorithm through such training. In fact, before taking this course, I don't know that the cumulative histograms are increasing linearly after implementing the histogram equalization. My experiment result is also consistent with this theory. We can see that a more uniform histogram generally corresponds to better overall contrast of the image.

- Median filter vs Adaptive median filter

The experiment results of this part is the strangest one I think, because the performance of median filter is better than that of adaptive median filter by using my image I took. I also tried to use other images for testing, even different file extensions, and the results were the same. But I also used the sample image of textbook to test my code and it seemed to work fine.

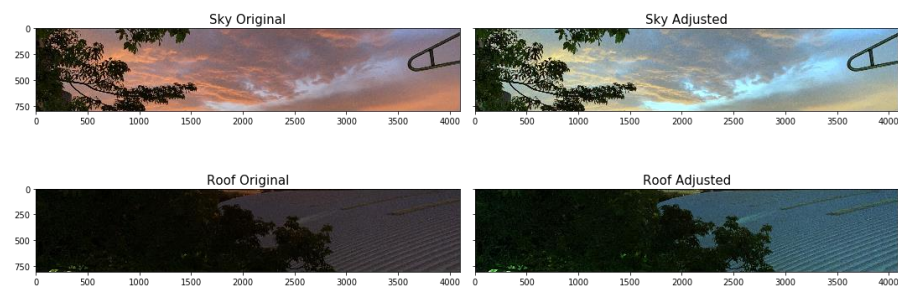
On the other hand, because the adaptive median filter can deal with impulsive noise with high probability, I have thought about whether the probability of noise added by my defined function is so low that it cannot display its performance on this task. However, this problem is still not resolved after I modify my code to increase the probability of noise. Additionally, I observed that if the probability of noise in the image is low, the adaptive median filter can use a smaller window size to improve the calculation speed. Conversely, if the probability of noise is high, the window size of the filter needs to be increased, to improve the filtering effect.

The only explanation I can think of is that this kind of the image processing method very depends on the selected images. I'm not sure I can say that this method doesn't adapt properly to all image data, that is, its generalization is not very well. Maybe more research can be done in the future to improve this problem.



- Color corrections

After this assignment, I think this is the most interesting one of the methods I have used, it has many applications that can be tried, and can extend a lot of style transfer issues. In fact, I was not very satisfied with the last result of the experiment part above, so I tried another way to make the image look more natural. Let's review my final result (the one with six images) here, we see that the red overcast was greatly diminished. However, we can see that there is a noticeable difference between the effectivity of our methodology in the top half and bottom half of the image. The sky portion of the image exhibits overexposure past the 95th percentile. Unfortunately, the roof only loses its red overcast percentiles lower than 95. Because the machine actually reads the image a matrix with many numbers, as such as I can easily split it and apply different parameters to each area. In this case, I split the image between the sky and the roof.



Finally, I combined the two parts using the concatenate function of NumPy. It looks like another different style now. The idea of this operation originated from the recent research of multimodal tasks by seniors of my lab.



- Morphological image processing

Morphology is the subject of the most recent self-study material, and it was the first time I heard about this technique, so I wanted to try

it out. It only needs to implement the two most widely used operations, Erosion and Dilation, the other operations are very simple.

After I actually executed my code, I verified that the opening operation will remove the smaller textures or noise, but also accidentally remove some important features. In contrast, the closing operation can retain important features, but there is no way to completely remove the noise.

Due to the aforementioned reasons, I do my best to try the following experiment, which is to improve the noise removal performance of the closing operation in the background. The method I think of is to change the kernel, and it was successful in removing the background noise. But in contrast, he retains fewer features on the fingerprint, resulting in more gaps. As a result, I don't have a clue how to deal with this at the moment.

- Remaining questions
  - Why did the adaptive median filter perform poorly in my second experiment?  
Need more experiments to define the problems and survey the solutions.
  - For the color correction of image, is there an algorithm that will be able to do most of the color correcting for more images?  
I think adaptive methods would be a good direction. Adaptively adjust the three color channels from the read image to let the resulting image looks realistic.
  - For morphological image processing, the method of material mentioned,  $(A \circ B) \cdot B$ , maybe has a better performance, but it still has some gaps in the image, is there a better solution or algorithm now?

```

import os
import cv2
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow.keras.preprocessing import image
from skimage.io import imshow, imread
from skimage import img_as_ubyte

def read_grayscale_image(path, filename):
    image = Image.open(path + filename)
    gray_image = image.convert("L")
    return gray_image

#----- Histogram Equalization -----
def display_hist(filename1, filename2):

    chest_xray_original = cv2.imread(filename1, cv2.IMREAD_GRAYSCALE)
    chest_xray_histeq = cv2.imread(filename2, cv2.IMREAD_GRAYSCALE)

    img_array_original = np.asarray(chest_xray_original)
    img_array_original = img_array_original.astype('int64')

    chest_xray_histeq = np.asarray(chest_xray_histeq)
    chest_xray_histeq = chest_xray_histeq.astype('int64')

    histarr_original = np.bincount(img_array_original.flatten(), minlength=256)
    histarr_histeq = np.bincount(chest_xray_histeq.flatten(), minlength=256)

    num_pixels_ori = np.sum(histarr_original)
    histarr_original = histarr_original/num_pixels_ori

    num_pixels_histeq = np.sum(histarr_histeq)
    histarr_histeq = histarr_histeq/num_pixels_histeq

```

```
plt.plot(histarr_original, color="royalblue", label="Original")
plt.plot(histarr_histeq, color="darkorange", label="Equalized")
plt.title('Normalized histogram', fontsize=16)
plt.xlabel('Pixel intensity', fontsize=14)
plt.ylabel('Distribution', fontsize=14)
plt.legend()
plt.show()
```

```
def cumul_hist(filename1, filename2):
```

```
    img_original = imread(filename1)
    img_histeq = imread(filename2)
```

```
    cumul_hist_original = np.zeros((256,))
    cumul_hist_eq = np.zeros((256,))
```

```
    c = 0
```

```
    for v in range(256):
```

```
        c += (img_original==v).sum()
```

```
        cumul_hist_original[v] = c
```

```
    cumul_hist_original /= cumul_hist_original.max()
```

```
    c = 0
```

```
    for v in range(256):
```

```
        c += (img_histeq==v).sum()
```

```
        cumul_hist_eq[v] = c
```

```
    cumul_hist_eq /= cumul_hist_eq.max()
```

```
plt.figure()
```

```
plt.plot(cumul_hist_original, color="royalblue", label="Original")
```

```
plt.plot(cumul_hist_eq, color="darkorange", label="Equalized")
```

```
plt.title('Normalized cumulative histogram', fontsize=16)
```

```
plt.xlabel('Pixel intensity', fontsize=14)
```

```
plt.ylabel('Distribution', fontsize=14)
```

```
plt.legend()
```

```
plt.show()
```

```

def histogram_equalization(image, save_filename):

    # convert to NumPy array

    img_array = np.asarray(image)
    img_array = img_array.astype('int64')

    ## Normalized cumulative histogram

    #flatten image array and calculate histogram via binning
    histogram_array = np.bincount(img_array.flatten(), minlength=256)

    #normalize
    num_pixels = np.sum(histogram_array)
    histogram_array = histogram_array/num_pixels

    #normalized cumulative histogram
    chistogram_array = np.cumsum(histogram_array)

    ## Pixel mapping lookup table
    transform_map = np.floor(255 * chistogram_array).astype(np.uint8)

    ## Transformation
    # flatten image array into 1D list
    img_list = list(img_array.flatten())

    # transform pixel values to equalize
    eq_img_list = [transform_map[p] for p in img_list]

    # reshape and write back into img_array
    eq_img_array = np.reshape(np.asarray(eq_img_list), img_array.shape)

    ##
    #convert NumPy array to pillow Image and write to file
    eq_img = Image.fromarray(eq_img_array, mode='L')
    plt.imshow(eq_img, cmap='gray')
    eq_img.save(save_filename)

```

```

path = "./images/"
chest_xray = read_grayscale_image(path, 'chest-xray.jpeg')

histogram_equalization(chest_xray, './images/chest-xray_hist.jpeg')

display_hist("./images/chest-xray.jpeg", "./images/chest-xray_hist.jpeg")
cumul_hist('./images/chest-xray.jpeg', './images/chest-xray_hist.jpeg')

#----- Median filter and Adaptive median filter -----
# Salt-and-Pepper noise
def sp_noise(image, prob):

    output = np.zeros(image.shape, np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]
    return output

image = cv2.imread('./images/cat.jpg', cv2.IMREAD_COLOR)
noise_img = sp_noise(image, 0.1)
cv2.imwrite('./images/cat_noise_sp.jpg', noise_img)

def filters(img):

    hImg = img.shape[0]
    wImg = img.shape[1]

    smax = 7
    m, n = smax, smax

```



```

# filling image edge
hPad = int((m-1) / 2)
wPad = int((n-1) / 2)
imgPad = np.pad(img.copy(), ((hPad, m-hPad-1), (wPad, n-wPad-1)),
mode="edge")

```

```

# Median filter
imgMedianFilter = np.zeros(img.shape)

```

```

# Adaptive median filter
imgAdaMedFilter = np.zeros(img.shape)

```

```

for i in range(hPad, hPad+hImg):
    for j in range(wPad, wPad+wImg):

```

```

        # Median filter
        ksize = 3
        k = int(ksize/2)
        pad = imgPad[i-k:i+k+1, j-k:j+k+1]
        imgMedianFilter[i-hPad, j-wPad] = np.median(pad)

```

```

        # Adaptive median filter
        ksize = 3
        k = int(ksize/2)
        pad = imgPad[i-k:i+k+1, j-k:j+k+1]
        zxy = img[i-hPad][j-wPad]
        zmin = np.min(pad)
        zmed = np.median(pad)
        zmax = np.max(pad)

```

```

        if zmin < zmed < zmax:
            if zmin < zxy < zmax:
                imgAdaMedFilter[i-hPad, j-wPad] = zxy
            else:
                imgAdaMedFilter[i-hPad, j-wPad] = zmed
        else:
            while True:
                ksize = ksize + 2

```

```

        if zmin < zmed < zmax or ksize > smax:
            break
        k = int(ksize / 2)
        pad = imgPad[i-k:i+k+1, j-k:j+k+1]
        zmed = np.median(pad)
        zmin = np.min(pad)
        zmax = np.max(pad)
    if zmin < zmed < zmax or ksize > smax:
        if zmin < zxy < zmax:
            imgAdaMedFilter[i-hPad, j-wPad] = zxy
        else:
            imgAdaMedFilter[i-hPad, j-wPad] = zmed

```

```

plt.figure(figsize=(9, 6))
plt.subplot(131), plt.axis('off'), plt.title("Original")
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.subplot(132), plt.axis('off'), plt.title("Median filter")
plt.imshow(imgMedianFilter, cmap='gray', vmin=0, vmax=255)
plt.subplot(133), plt.axis('off'), plt.title("Adaptive median filter")
plt.imshow(imgAdaMedFilter, cmap='gray', vmin=0, vmax=255)
plt.tight_layout()
plt.show()

```

```

img_cat = cv2.imread("./images/cat_noise_sp_gray.jpg",
cv2.IMREAD_GRAYSCALE)
filters(img)
img_sample = cv2.imread("./images/book_example.tif",
cv2.IMREAD_GRAYSCALE)
filters(img_sample)

```

#----- Color correction -----

```

def channel_statistics(image):
    df_color = []
    for i in range(0, 3):
        max_color = np.max(image[:, :, i])
        mean_color = np.mean(image[:, :, i])
        median_color = np.median(image[:, :, i])

```

```

perc_90 = np.percentile(image[:, :, i], 90, axis=(0,1))
perc_95 = np.percentile(image[:, :, i], 95, axis=(0,1))
perc_99 = np.percentile(image[:, :, i], 99, axis=(0,1))

row = (max_color, mean_color, median_color,
       perc_90, perc_95, perc_99)
df_color.append(row)

return pd.DataFrame(df_color,
                    index = ['Red', 'Green', 'Blue'],
                    columns = ['Max', 'Mean', 'Median',
                              'P_90', 'P_95', 'P_99'])

def mean_and_median_adjustment(image):
    fig, ax = plt.subplots(2, 2, figsize=(10, 7), sharey = True)
    f_size = 16
    ax[0][0].imshow(image)
    ax[0][0].set_title('Original', fontsize = f_size)
    ax[0][1].imshow(img_as_ubyte((image / np.mean(image)).clip(0, 1)))
    ax[0][1].set_title('Mean Adjusted', fontsize = f_size)

    ax[1][0].imshow(img_as_ubyte((image / np.median(image)).clip(0, 1)))
    ax[1][0].set_title('Median Adjusted', fontsize = f_size)
    ax[1][1].imshow(img_as_ubyte((image / np.max(image)).clip(0, 1)))
    ax[1][1].set_title('Max Adjusted', fontsize = f_size);

    fig.tight_layout()

mean_and_median_adjustment(image_scene)

def redmax_adjustment(image):

    fig, ax = plt.subplots(1, 3, figsize=(15,7), sharey = True)
    f_size = 15
    ax[0].imshow(image)
    ax[0].set_title('Original', fontsize = f_size)
    ax[1].imshow(img_as_ubyte((image /

```

```

        [np.max(image[:, :, 0]),
         np.mean(image[:, :, 1]),
         np.mean(image[:, :, 2])])
        .clip(0, 1)))
ax[1].set_title('Red : Max, Others : Mean', fontsize = f_size)
ax[2].imshow(img_as_ubyte((image /
        [np.max(image[:, :, 0]),
         np.median(image[:, :, 1]),
         np.median(image[:, :, 2])])
        .clip(0, 1)))
ax[2].set_title('Red : Max, Others : Median', fontsize = f_size)

redmax_adjustment(image_scene)

def percentile_adjustment(image):

    fig, ax = plt.subplots(2, 3, figsize=(20, 10), sharey = True)
    f_size = 15
    red = 99.75
    parameter_matrix = [[red] + [99]*3,
                        [red] + [95]*3,
                        [red] + [90]*3,
                        [red] + [85]*3,
                        [red] + [80]*3]

    ax[0][0].imshow(image)
    ax[0][0].set_title('Original', fontsize = f_size)
    ax[0][1].imshow(img_as_ubyte((image / [np.percentile(image[:, :, i],
parameter_matrix[0][i], axis=(0, 1))
        for i in range (0, 3)]).clip(0, 1)))

    ax[0][1].set_title('Red :' + str(red) + ', Others :' + str(parameter_matrix[0][1]),
    fontsize = f_size)
    ax[0][2].imshow(img_as_ubyte((image / [np.percentile(image[:, :, i],
parameter_matrix[1][i], axis=(0, 1))
        for i in range (0, 3)]).clip(0, 1)))

```

```

ax[0][2].set_title('Red :' + str(red) + ', Others :' + str(parameter_matrix[1][1]),
fontsize = f_size)
ax[1][0].imshow(img_as_ubyte((image / [np.percentile(image[:, :, i],
parameter_matrix[2][i], axis=(0, 1))
for i in range (0, 3))].clip(0, 1)))

```

```

ax[1][0].set_title('Red :' + str(red) + ', Others :' + str(parameter_matrix[2][1]),
fontsize = f_size)
ax[1][1].imshow(img_as_ubyte((image / [np.percentile(image[:, :, i],
parameter_matrix[3][i], axis=(0, 1))
for i in range (0, 3))].clip(0, 1)))

```

```

ax[1][1].set_title('Red :' + str(red) + ', Others :' + str(parameter_matrix[3][1]),
fontsize = f_size)
ax[1][2].imshow(img_as_ubyte((image / [np.percentile(image[:, :, i],
parameter_matrix[4][i], axis=(0, 1))
for i in range (0, 3))].clip(0, 1)))

```

```

ax[1][2].set_title('Red :' + str(red) + ', Others :' + str(parameter_matrix[4][1]),
fontsize = f_size);

```

```

percentile_adjustment(image_scene)

```

```

def seperate_params(image):

```

```

    f_size = 15
    parameter_matrix = [[99.9] + [97.75]*3, [99.75] + [92]*3]
    fig, ax = plt.subplots(2, 2, figsize=(15,6), sharey = True)
    ax[0][0].imshow(image[0:1500])
    ax[0][0].set_title('Sky Original', fontsize = f_size)
    ax[1][0].imshow(image[1500:])
    ax[1][0].set_title('Roof Original', fontsize = f_size)
    scene_correction_top = img_as_ubyte(((image[0:1500]/
[ np.percentile(image[:, :, i], parameter_matrix[0][i], axis=(0, 1)) for i in
range(3) ] ).clip(0,1)))
    ax[0][1].imshow(scene_correction_top)
    ax[0][1].set_title('Sky Adjusted', fontsize = f_size)

```

```

        scene_correction_bottom = img_as_ubyte(((image[1500:]/
[ np.percentile(image[:, :, i], parameter_matrix[1][i], axis=(0, 1)) for i in
range(3)]) .clip(0,1)))
        ax[1][1].imshow(scene_correction_bottom)
        ax[1][1].set_title('Roof Adjusted', fontsize = f_size);
        fig.tight_layout()

    return scene_correction_top, scene_correction_bottom

```

```

scene_correction_top, scene_correction_bottom = seperate_params(image_scene)

```

```

imshow(np.concatenate((scene_correction_top, scene_correction_bottom), axis=0))

```

```

#----- Morphological image processing -----

```

```

def erosion(img1):

```

```

    m, n= img1.shape

```

```

    #plt.imshow(img1, cmap="gray")

```

```

    #plt.figure()

```

```

    # Define the structuring element

```

```

    k = 3 # different sizes of the structuring element

```

```

    SE = np.ones((k, k), dtype=np.uint8)

```

```

    constant = (k-1)//2

```

```

    # Define new image to store the pixels of eroded image

```

```

    imgErode = np.zeros((m,n), dtype=np.uint8)

```

```

    # Erosion

```

```

    for i in range(constant, m-constant):

```

```

        for j in range(constant, n-constant):

```

```

            temp = img1[i-constant:i+constant+1, j-constant:j+constant+1]

```

```

            product = temp*SE

```

```

            imgErode[i,j] = np.min(product)

```

```

    plt.imshow(imgErode, cmap="gray")

```



```

cv2.imwrite("./images/eroded_finger.png", imgErode)

img_original_finger = cv2.imread("./images/noisy_fingerprint.tif", 0)
erosion(img_original_finger)

def dilation(img1):

    p, q = img1.shape

    #plt.imshow(img1, cmap="gray")
    #plt.figure()

    # Define new image to store the pixels of dilated image
    imgDilate = np.zeros((p,q), dtype=np.uint8)

    # Define the structuring element
    SED = np.array([[0,1,0], [1,1,1], [0,1,0]])
    constant1 = 1

    # Dilation
    for i in range(constant1, p-constant1):
        for j in range(constant1, q-constant1):
            temp = img1[i-constant1:i+constant1+1, j-constant1:j+constant1+1]
            product = temp*SED
            imgDilate[i,j] = np.max(product)

    plt.imshow(imgDilate, cmap = "gray")
    cv2.imwrite("./images/dilated_finger.png", imgDilate)

dilation(img_original_finger)

def opening(img1):

    m, n= img1.shape

    #plt.imshow(img1, cmap="gray")

```

```

plt.figure()

# Define the structuring element (erosion)
k = 3 # different sizes of the structuring element
SE = np.ones((k,k), dtype=np.uint8)
constant = (k-1)//2

# Define the structuring element (dilation)
SED = np.array([[0,1,0], [1,1,1], [0,1,0]])
constant1 = 1

# Define new image to store the pixels of eroded image
imgErode = np.zeros((m,n), dtype=np.uint8)
# Define new image to store the pixels of opening image
imgOpening = np.zeros((m,n), dtype=np.uint8)

# Erosion
for i in range(constant, m-constant):
    for j in range(constant, n-constant):
        temp = img1[i-constant:i+constant+1, j-constant:j+constant+1]
        product = temp*SE
        imgErode[i,j] = np.min(product)

# Dilation
for i in range(constant1, m-constant1):
    for j in range(constant1, n-constant1):
        temp = imgErode[i-constant1:i+constant1+1, j-constant1:j+constant1+1]
        product = temp*SED
        imgOpening[i,j] = np.max(product)

plt.imshow(imgOpening, cmap = "gray")
cv2.imwrite("./images/opening_finger.png", imgOpening)

opening(img_original_finger)

```

```

def closing(img1):

    m, n= img1.shape

    #plt.imshow(img1, cmap="gray")
    #plt.figure()

    # Define the structuring element (erosion)
    k = 3 # different sizes of the structuring element
    SE = np.ones((k,k), dtype=np.uint8)
    constant = (k-1)//2

    # Define the structuring element (dilation)
    SED = np.array([[0,1,0], [1,1,1], [0,1,0]])
    constant1 = 1

    # Define new image to store the pixels of dilated image
    imgDilate = np.zeros((m,n), dtype=np.uint8)
    # Define new image to store the pixels of closing image
    imgClosing = np.zeros((m,n), dtype=np.uint8)

    # Dilation
    for i in range(constant1, m-constant1):
        for j in range(constant1, n-constant1):
            temp = img1[i-constant1:i+constant1+1, j-constant1:j+constant1+1]
            product = temp*SED
            imgDilate[i,j] = np.max(product)

    # Erosion
    for i in range(constant, m-constant):
        for j in range(constant, n-constant):
            temp = imgDilate[i-constant:i+constant+1, j-constant:j+constant+1]
            product = temp*SE
            imgClosing[i,j] = np.min(product)

    plt.imshow(imgClosing, cmap = "gray")
    cv2.imwrite("./images/closing_finger.png", imgClosing)

```

```
closing(img_original_finger)
```

```
def gradient(img1):
```

```
    m, n= img1.shape
```

```
    #plt.imshow(img1, cmap="gray")
```

```
    #plt.figure()
```

```
    # Define the structuring element (erosion)
```

```
    k = 3 # different sizes of the structuring element
```

```
    SE = np.ones((k,k), dtype=np.uint8)
```

```
    constant = (k-1)//2
```

```
    # Define the structuring element (dilation)
```

```
    SED = np.array([[0,1,0], [1,1,1], [0,1,0]])
```

```
    constant1 = 1
```

```
    # Define new image to store the pixels of eroded image
```

```
    imgErode = np.zeros((m,n), dtype=np.uint8)
```

```
    # Define new image to store the pixels of dilated image
```

```
    imgDilate = np.zeros((m,n), dtype=np.uint8)
```

```
    # Define new image to store the pixels of gradient image
```

```
    imgGradient = np.zeros((m,n), dtype=np.uint8)
```

```
    # Erosion
```

```
    for i in range(constant, m-constant):
```

```
        for j in range(constant, n-constant):
```

```
            temp = img1[i-constant:i+constant+1, j-constant:j+constant+1]
```

```
            product = temp*SE
```

```
            imgErode[i,j] = np.min(product)
```

```
    # Dilation
```

```
    for i in range(constant1, m-constant1):
```

```
        for j in range(constant1, n-constant1):
```

```
            temp = img1[i-constant1:i+constant1+1, j-constant1:j+constant1+1]
```

```
            product = temp*SED
```

```
imgDilate[i,j] = np.max(product)
```

```
imgGradient = imgErode - imgDilate
```

```
plt.imshow(imgGradient, cmap = "gray")
```

```
cv2.imwrite("./images/gradient_finger.png", imgGradient)
```

```
gradient(img_original_finger)
```