CS484
Homework Assignment 1

Boran Kılıç
22103444

Morphological Operations

In this question it is asked to write our custom morphological operations functions and apply morphological operations to the provided image to get a desired output.

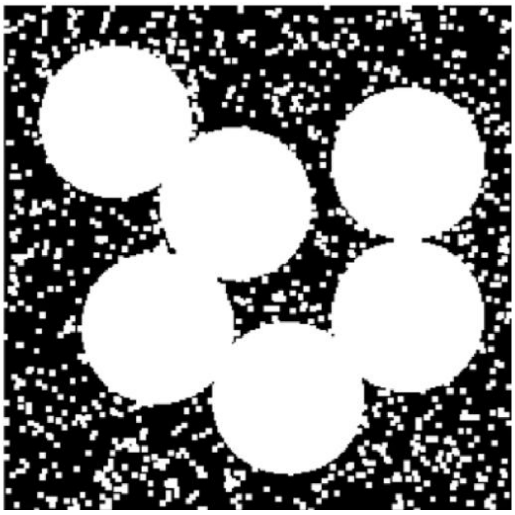


Fig.1 Provided image

It was asked in the question to remove the noise in the background of the image so that only the circles at the middle are white and the background is completely black. After the functions are written, shape of the structuring element is decided. Since the we have circles in the provided image it was beneficial to have a structuring element that has a shape of a circle. A circle shaped 9x9 structuring element is designed as follows.

```
se=[0 0 0 0 1 0 0 0 0
    0 0 1 1 1 1 1 0 0
    0 1 1 1 1 1 1 1 0
    0 1 1 1 1 1 1 1 0
    1 1 1 1 1 1 1 1 1
    0 1 1 1 1 1 1 1 0
    0 1 1 1 1 1 1 1 0
    0 0 1 1 1 1 1 0 0
    0 0 0 0 1 0 0 0 0];
```

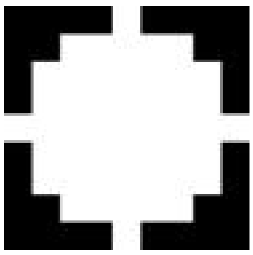


Fig.2 a-b Structuring element

After the structuring element is designed the sequence of morphological operations is decided. First operation was decided as erosion because erosion shrinks the connected sets of 1s of a binary image. It can be used for getting rid of the white pixels in the background. Erosion is applied 10 times until white pixels in the background are completely removed. The reason for that is the structuring element used in the operation was relatively small. After that, of course the big white circles are also got smaller. In order to make them bigger again dilation is applied because dilation expands the connected sets of 1s of a binary image. It can be used for growing features. Again the dilation is applied 10 times because of the small structuring element size. In the end the following result is obtained.

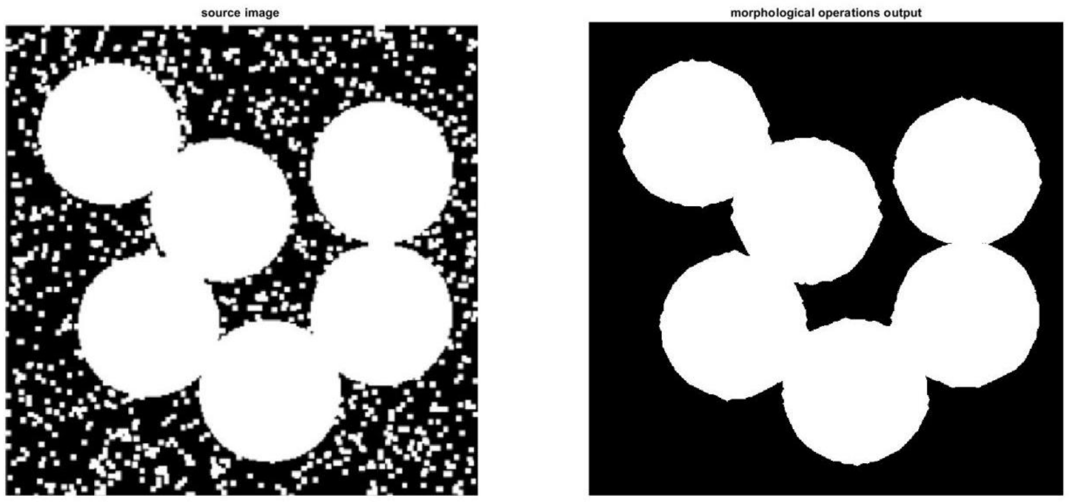


Fig.3 Result of the morphological operations sequence compared with the source image

In Fig.3 it is visible that the desired output is achieved. If the structuring element was bigger so that erosion and dilation are applied once in the same order that operation would be an equivalence of an ***opening*** operation applied once.

Histogram-Based Image Enhancement

In this part of the homework, implementing two functions is asked. One of which is *histogram* function and the other is *contrast_stretching* function.

Part 1

This part contains the implementation of the function that generates a histogram of a greyscale image. In order to complete this task an array named *hist_values* is created with *zeros(1 ,256)* command. Each location of this array corresponds to the frequency of an intensity level with the following relation:

location = intensity level +1

Because MATLAB indexing starts from 1.
After flowing through each pixel of the image the *hist_values* array is plotted using MATLAB's built-in *bar()* function.
The function is tested with the provided images and the following results are obtained.

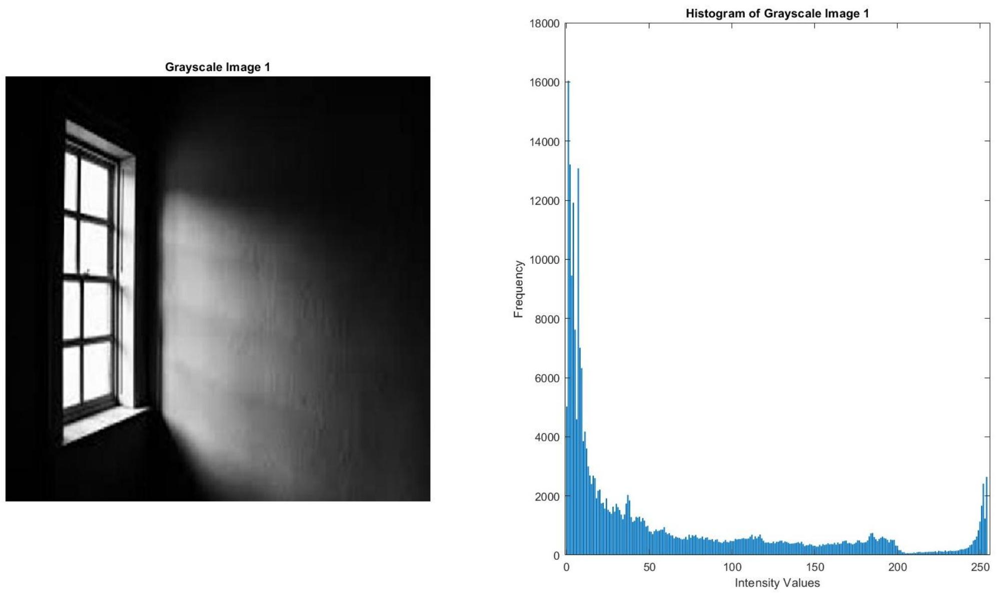


Fig.4 Histogram of the first grayscale image

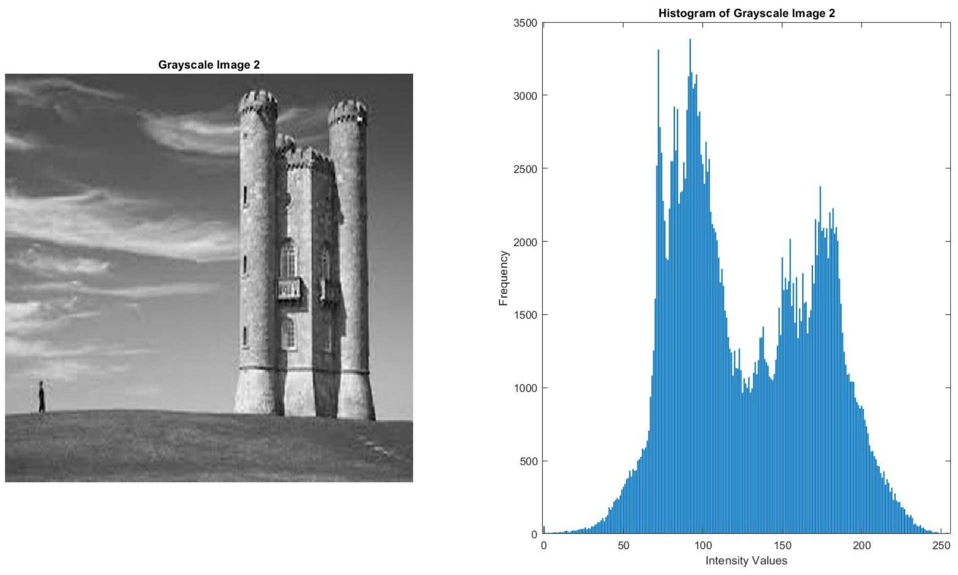


Fig.5 Histogram of the second grayscale image

The results were consistent with the MATLAB’s built-in histogram function.

Part 2

In this part the task was to write a function that does contrast stretching   to a given grayscale image. Contrast streching is a process that is applied in order to enhance the contrast of an image by distributing the intensity values of the pixels of the image in grayscale. This redistribution process has been done with the following formula:

$$f(x) = \frac{(x-a)}{(b-a)}(d-c)+c$$

In the original image, **x** represents the pixel intensity. The values **a** and **b** are the minimum and maximum pixel intensities in the input image, respectively, while **c** and **d** denote the desired minimum and maximum pixel intensity values in the output image, respectively.

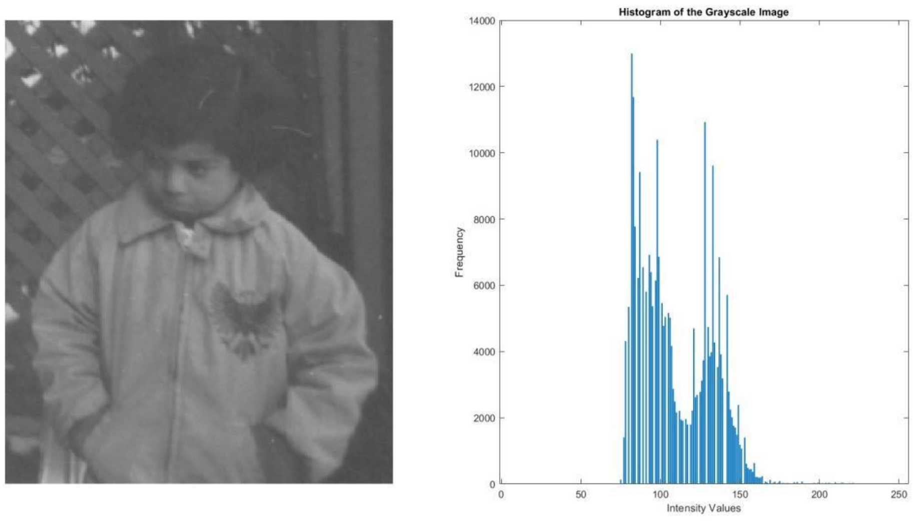


Fig.6 Original image and its distribution

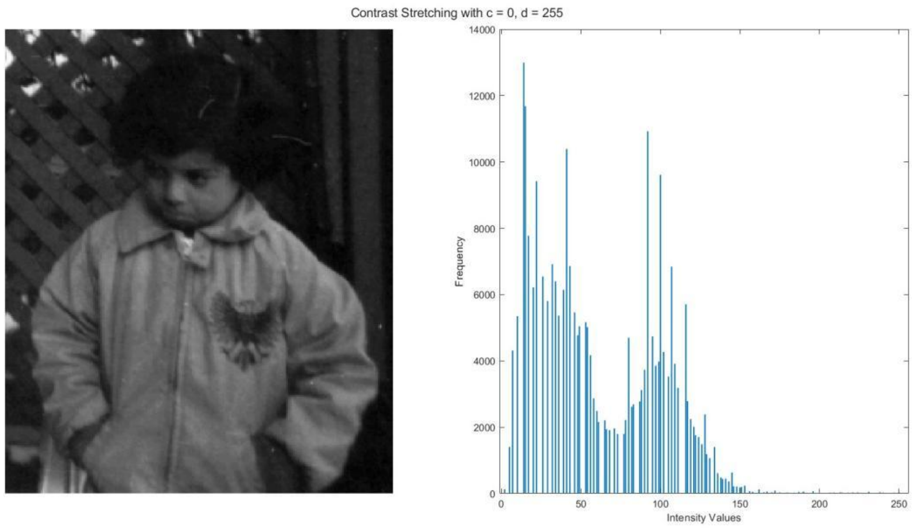


Fig.7 Contrast Streching with c=0 ,   d=255 and its distribution

As it can be seen from *Fig.6&7* the stretched image has higher contrast compared to the original image.

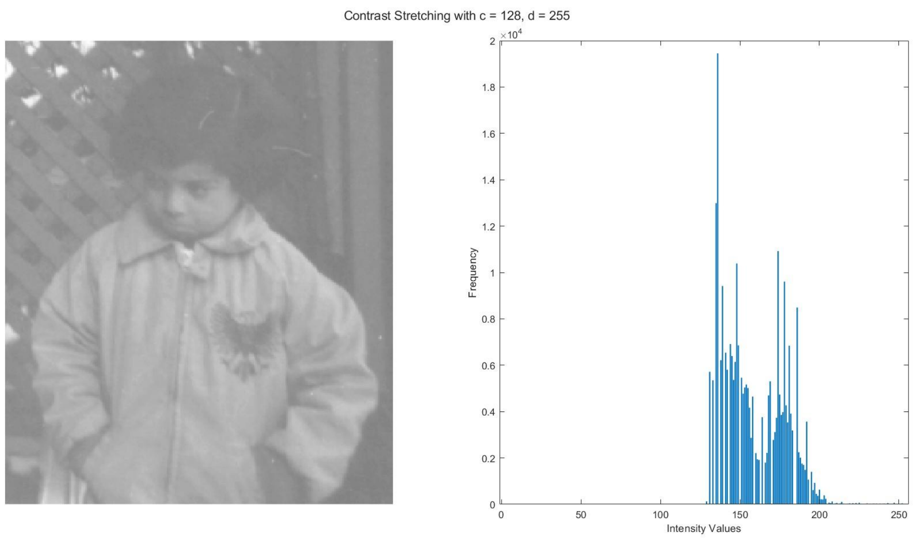


Fig.8 Contrast Streching with c=128 ,   d=255 and its distribution

In *Fig.8* the distribution of the pixel intensity values are fitted to the right half of the grayscale. Therefore, the resulting image is whiter (brighter) compared to the others.

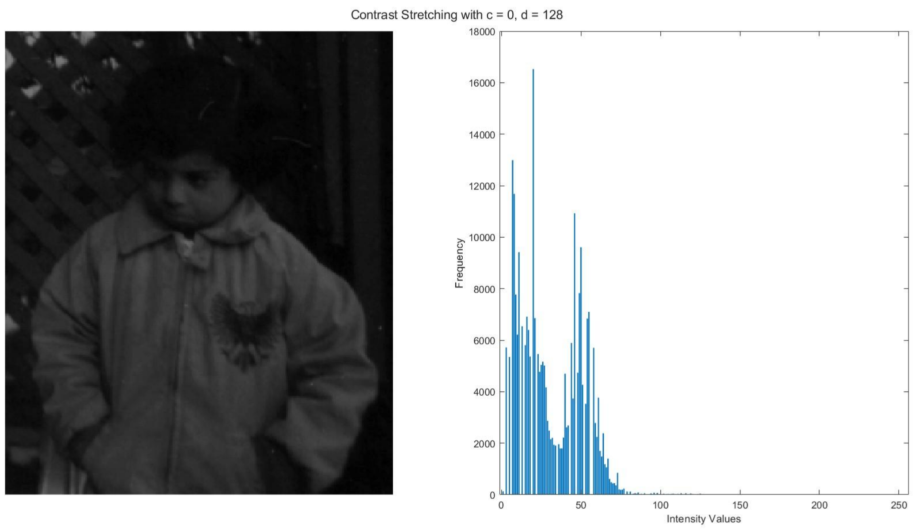


Fig.9 Contrast Streching with c=0 ,   d=128 and its distribution

In *Fig.9* the distribution of the pixel intensity values are fitted to the left half of the grayscale. Therefore, the resulting image is darker compared to the others.

To sum up, different [c,d] values determines the boundaries of the distribution in the grayscale which determines the color brightness and the contrast of the image. Higher [c,d] range results in a higher contrast.

Otsu Thresholding

In this part the task was to write a function named *otsu_threshold (source_image)* which implements thresholding using Otsu's method. Otsu Thresholding also known as Automatic Thresholding takes a grayscale image and converts it to a binary image. This conversion is done by determining a threshold from the distribution of the image and assigning 0 to the values below that threshold and 1 to the values above. Otsu's method determines this threshold as the value that minimizes the combined weighted variances within each of the two groups formed by dividing the grayscale levels at the threshold.

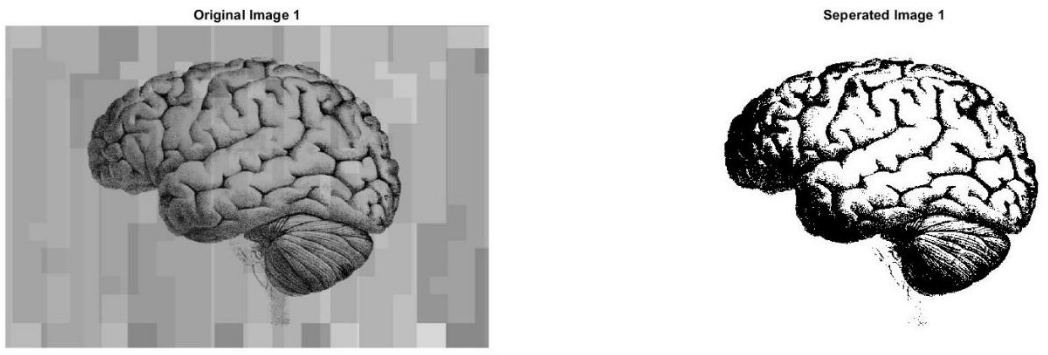


Fig.10 Otsu Thresholding applied to the first image (otsu_1.png)

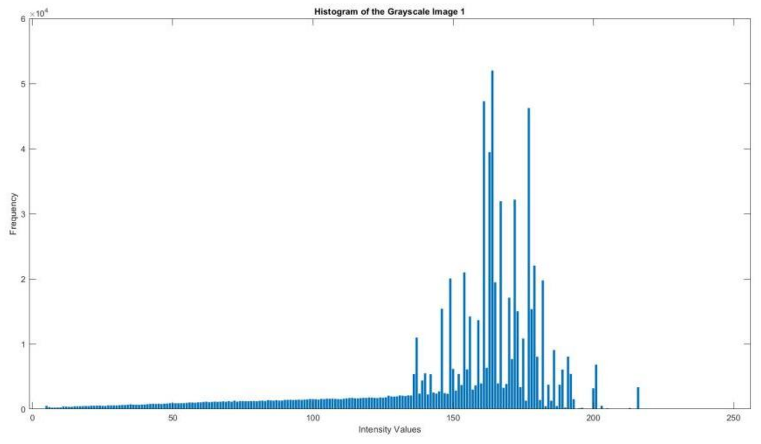


Fig.11 Histogram of the first image (otsu_1.png)

In *Fig.10* the original image has a brain image on the background and some noise on the foreground. Otsu tresholding enables to separate the brain image from the noisy foreground. However, some parts of the image in the background may be lost during the process for example in this case brainstem is not visible in the binary image because its colors are closer to the colors in the foreground which have been got rid of. A benefit of Otsu's thresholding is that it makes it easier to extract and image for object recognition at the same time it may loose some information during the process.

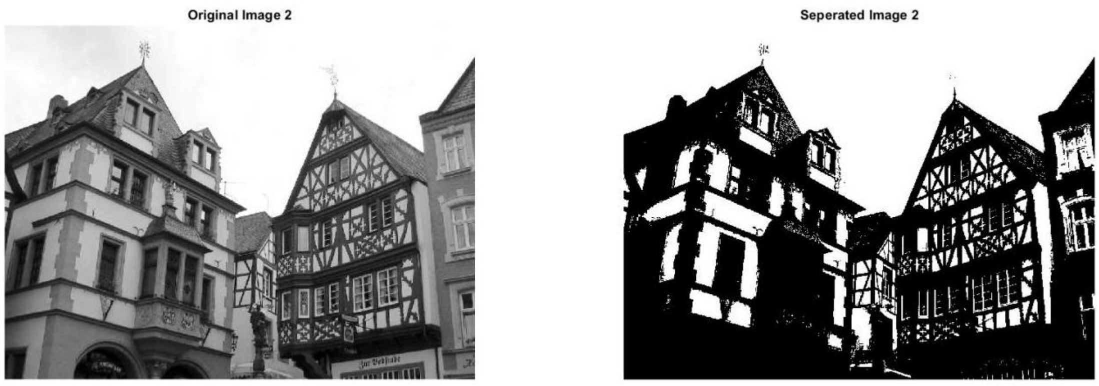


Fig.12 Otsu Thresholding applied to the second image (otsu_2.jpg)

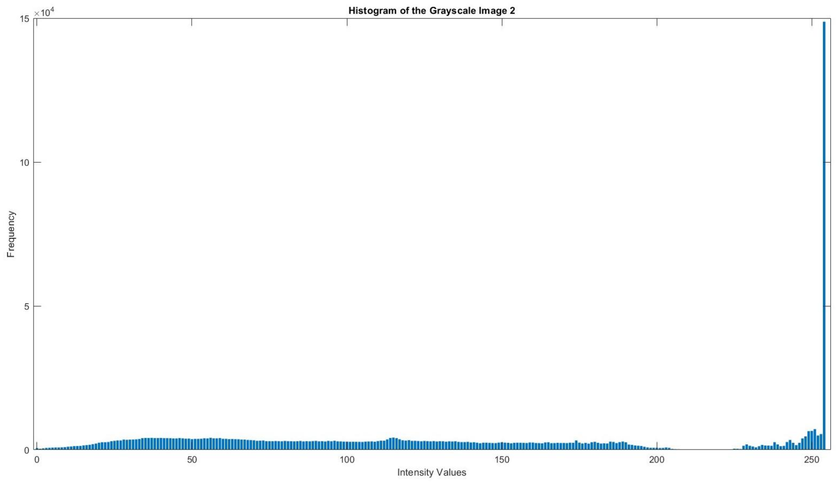
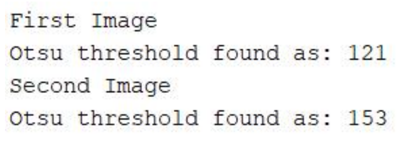


Fig.13 Histogram of the second image (otsu_2.jpg)

In *Fig.12* most of the details in the original image is lost this is due to weak edges. Since, the contrast of the objects edges are is minimal (for instance the windows), the chosen threshold has failed to isolate the details. Therefore, Otsu's method may not perform well when objects in the image have weak or blurry edges.



```
First Image
Otsu threshold found as: 121
Second Image
Otsu threshold found as: 153
```

Fig.14 Threshold values from the command window

Otsu's method assumes bimodal histogram however as it is visible in *Fig.11&13* both histograms are not bimodal therefore the results were not perfect.
In conclusion, Otsu's method is very beneficial tool to separate two layers of the images however it is not always gives perfect results as it loses some low contrast details.

2-D Convolution in Spatial and Frequency Domain

Part 1

In this part the task was to implement two-dimensional convolution in the spatial domain. First step was to write the *convolution* function which takes an image and a filter as 2D matrices.
After the *convolution* function had been written, next task was to implement edge detection using the function. For this purpose following image is given:

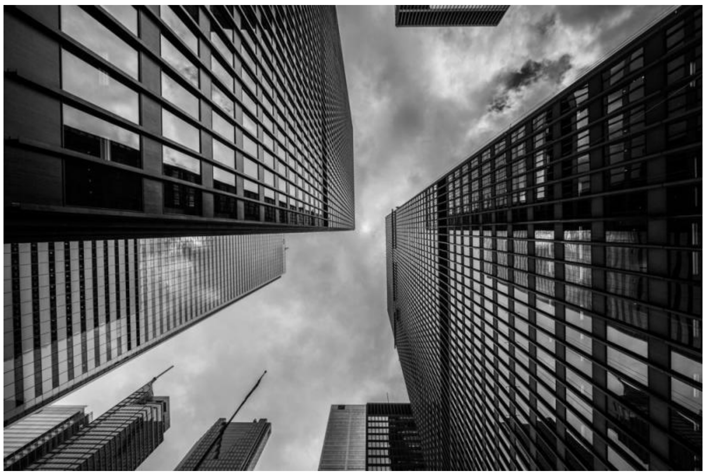
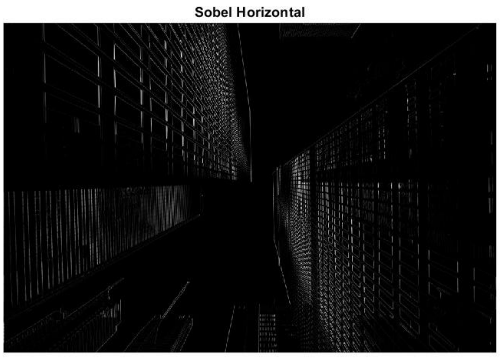


Fig.15 Image to apply edge detection

Edge detection is applied to the image in *Fig.15* using two types of filters: Sobel and Prewitt filters.
Sobel filter has the following form where Sobel_x is used for the detection of horizontal edges and Sobel_y is used for the detection of vertical edges:

$$\text{Sobel}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Sobel}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



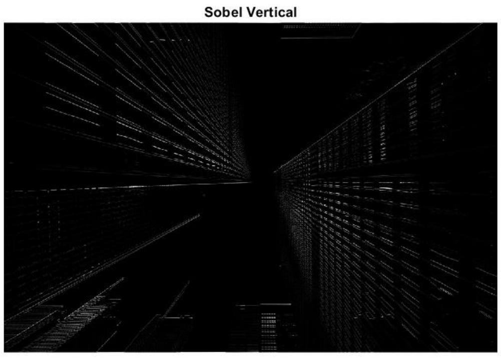


Fig.16 (a) Horizontal Sobel filter applied image *(b)Vertical Sobel filter applied image*

Fig.16 Illustrates the results after the images are convoluted with the filters and the element-wise squared. The reason after the square operation is applied in order to get the same results for the inversely and directly correlated edges. This way the edges are visualized better. By summing up the results of vertical and horizontal filters we get the following output that contains all the edges:

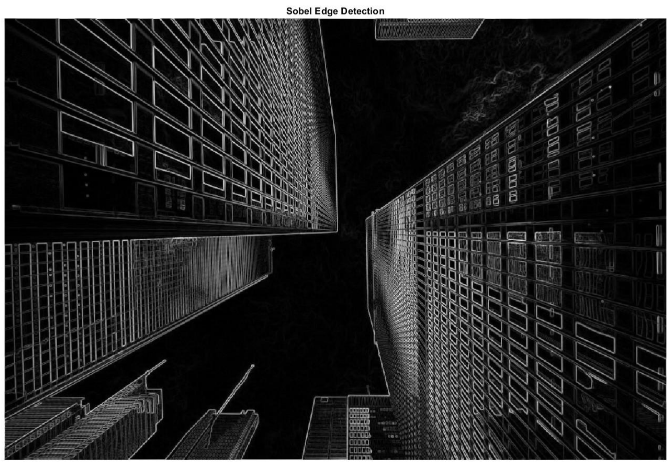
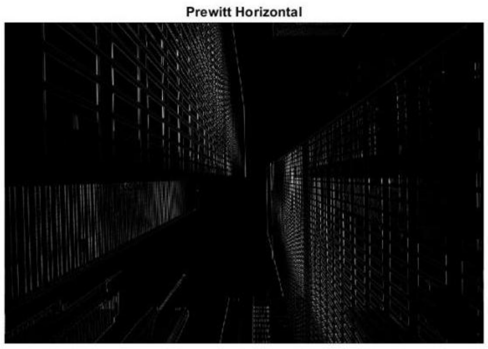


Fig.17 Edge detection with Sobel filter result

Similarly, Prewitt filter has the following form where Prewitt_x is used for the detection of horizontal edges and Prewitt_y is used for the detection of vertical edges:

$$\text{Prewitt}_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Prewitt}_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$



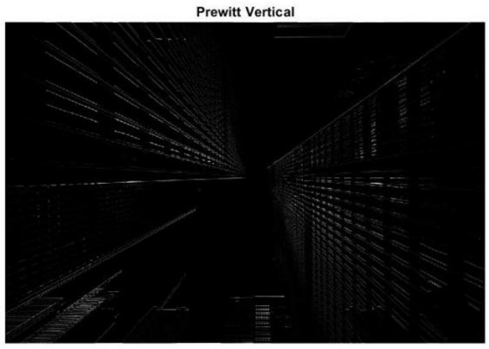


Fig.18 (a) Horizontal Prewitt filter applied image *(b)Vertical Prewitt filter applied image*

Following the same steps with the Sobel filter we have the following result:

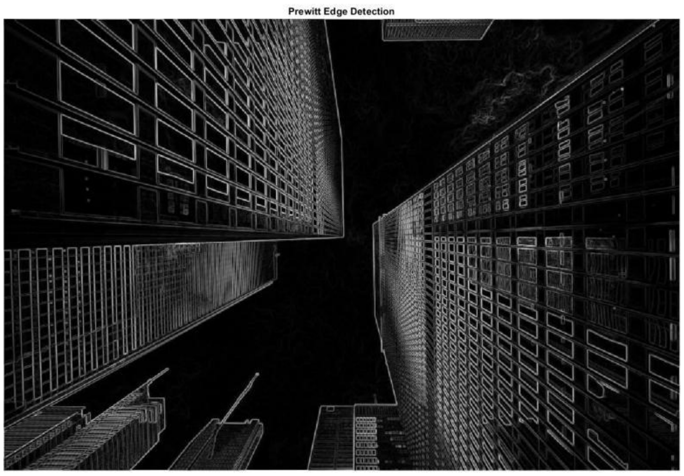


Fig.19 Edge detection with Prewitt filter result

As it can be seen from the *Fig.17&19* the filters are more focused on the high frequency components and high frequency components are usually located at the buildings in the image.

Part 2

In this part, the task was to apply a Gaussian low-pass filter to a grayscale image in frequency domain. The first step was to apply 2D Fourier transform to the image in order to go to frequency domain.The 2D Fourier Transform is taken as follows:

```
img_freq = fftshift(fft2(ifftshift(img)));
```

This code applies a 2D Fourier Transform to the **img** matrix, centering the frequency components by shifting the zero-frequency component to the center of the spectrum

Call magnitude spectrum of the original image, F. After that the Gaussian filter in frequency domain is defined as the following:

$$H = \exp\left(-\frac{u^2 + v^2}{\sigma^2}\right)$$

Where σ is the standard deviation of the filter which determines the bandwidth (radius) of the filter. σ is determined as 10 as this sufficiently illustrates the effect of the filter.
Since, according to convolution theorem, convolution in spatial domain corresponds to multiplication in the frequency domain, The filter is simply applied as the following:

$$Y = H x F$$

By going back to spatial domain by applying Inverse Fourier Transform we have the following result.

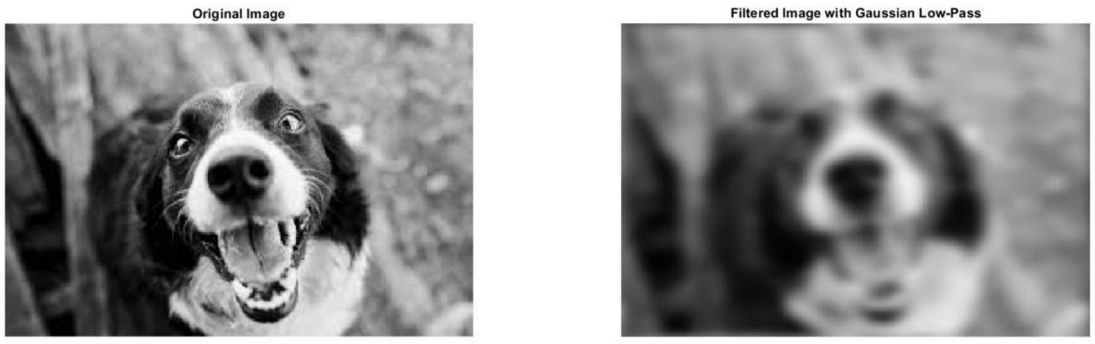


Fig.20 Gaussian low-pass filter result

The Inverse 2D Fourier Transform is taken with the following code:

```
filtered_img = fftshift(ifft2(ifftshift(filtered_freq)))
```

This code applies an inverse 2D Fourier Transform to the **filtered_freq** matrix, centering it first by shifting the zero-frequency component to the edges, and then shifts it back after transforming to produce the spatial domain image **filtered_img**.
It is visible in *Fig.20* that low-pass filter decreased the contrast of the image, hence we got a image is blurry.