CS 484
Spring 2023
Homework Assignment 1

Yiğit Ekin
21901784
Section 01

Q1)
In order to implement erosion and dilation methods, the following third party libraries have been used:

- **Numpy:** for converting the image to the matrix and applying faster operations on the matrix.
- **Matplotlib.pyplot:** in order to display the histogram.
- **PIL.Image:** in order to read the image.

After the implementation, several methods were asked to be applied in order to make Figure 1.1 more readable.

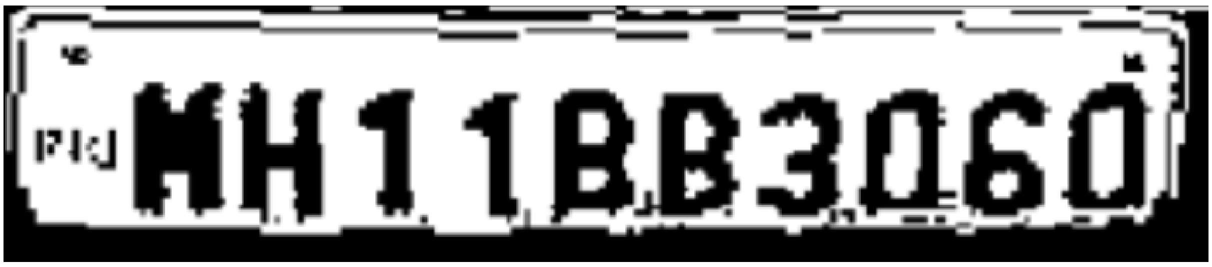


Figure 1.1: Image to apply morphological operations

In order to apply morphological operations, a structuring element is needed. For achieving such goal, the following element is used:

0	0	0
0	0	0
0	0	0

This element is preferred because we want to dilate or erose the black pixels. As a result, an binary image of all black pixels are selected for structuring element. In addition, an homogenous distribution is selected in order to conserve the overall structure.

First of all, erosion was applied in order to remove some of the unnecessary connected components. The result is the following:

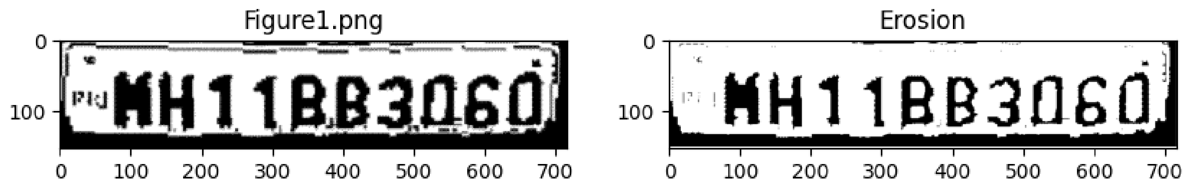


Figure 1.2: Figure1.png and its erosed version

After removing some of the connected components with erosion, opening was done on the image in order to remove “pkj” text and then later dilate the aimed image to display it in a more readable manner.

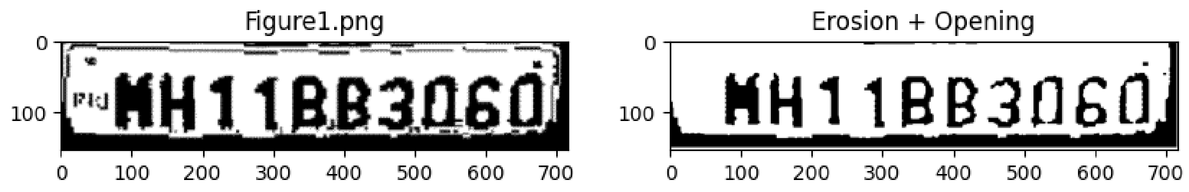


Figure 1.3: Figure1.png after erosion and opening

After that, closing was done on the image which is a dilation with erosion in a sequential manner. The aim was to maket he text more bigger and then remove the noise made by it from the dilation with erosion.

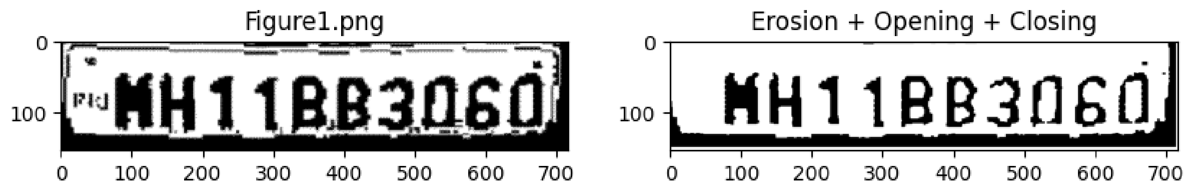


Figure 1.4: Figure1.png after erosion, opening and closing

Finally, in order to display the zeros in a more readable manner, dilation operation was done so that it is bigger.

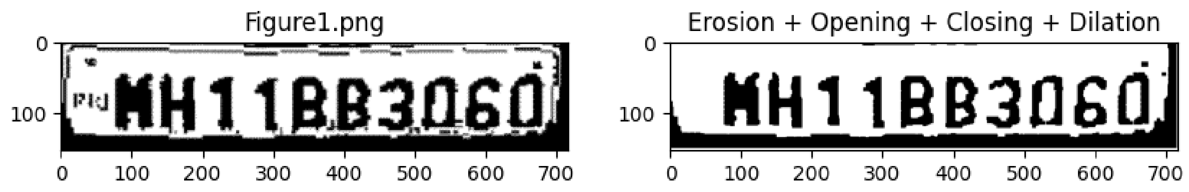
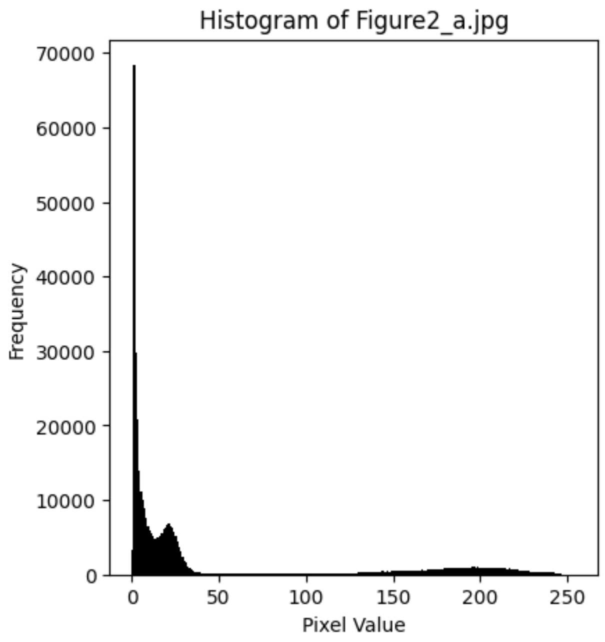


Figure 1.5: Figure1.png after erosion, opening, closing and dilation

Q2)
In order to implement histogram method, the following third party libraries have been used:

- **Numpy:** for converting the image to the matrix and applying faster operations on the matrix.
- **Matplotlib.pyplot:** in order to display the histogram.
- **PIL.Image:** in order to read the image.

The implemented method takes a grayscale image matrix and is feasible for any grayscale image. After the implementation, the generated histograms for the images are the following:



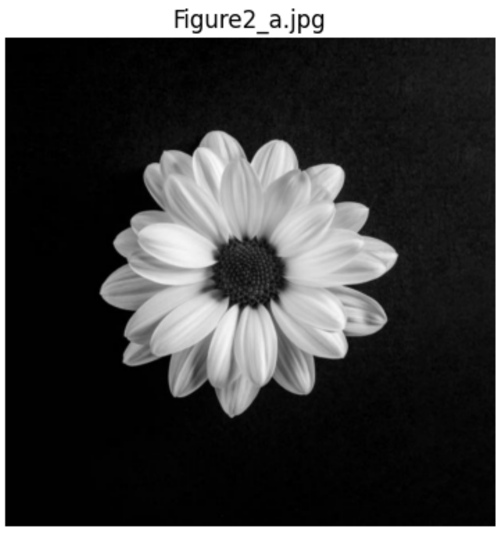
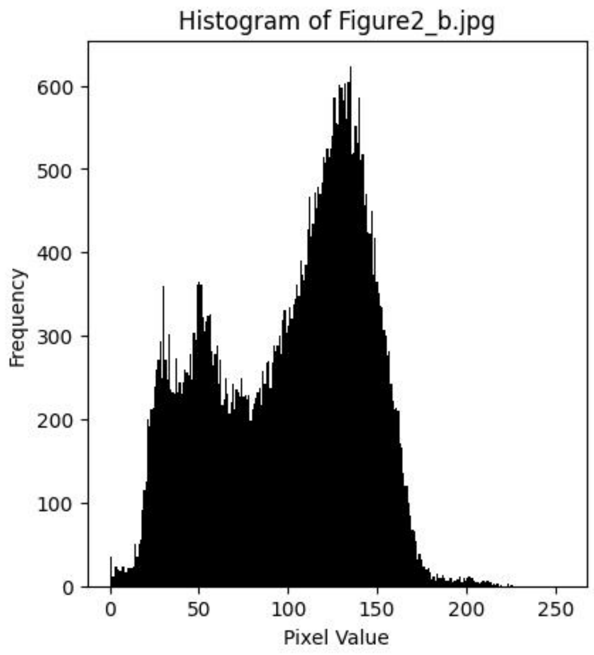


Figure 2.1: Generated histogram for figure2_b.jpg

As the image contains huge amount of very dark pixels in the background, the histogram has a sudden peak at the left side. Also, as the leaves of the flower is a bright white color, we can also see some peak at the near 200 values on the histogram. However, as the darkers outweight the brighters, the histogram is strong on the left side.



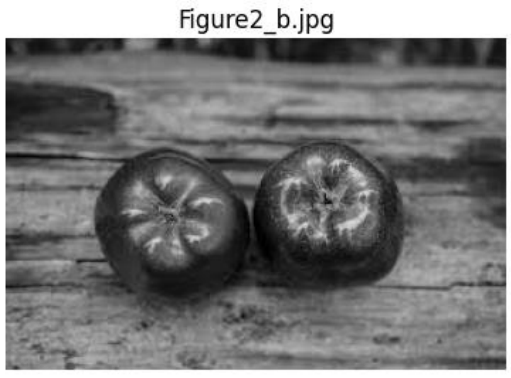


Figure 2.2: Generated histogram for figure2_b.jpg

As the image contains huge amount of dark-gray and light gray pixels in the both background and foreground, the histgram is more focused on the middle part of the spectrum. However, as there are less white pixels and more dark pixels (shadows on fruit and table) the darker region has also some higher intensity compared to right region (white) but lower compared to middle region.

Q3)
In order to implement otsu tresholding method, the following third party libraries have been used:

- **Numpy:** for converting the image to the matrix and applying faster operations on the matrix.
- **Matplotlib.pyplot:** in order to display the histogram.
- **PIL.Image:** in order to read the image.

The implemented method takes a grayscale image matrix and is feasible for any grayscale image. After the implementation, the generated binary images with otsu tresholding are the following:

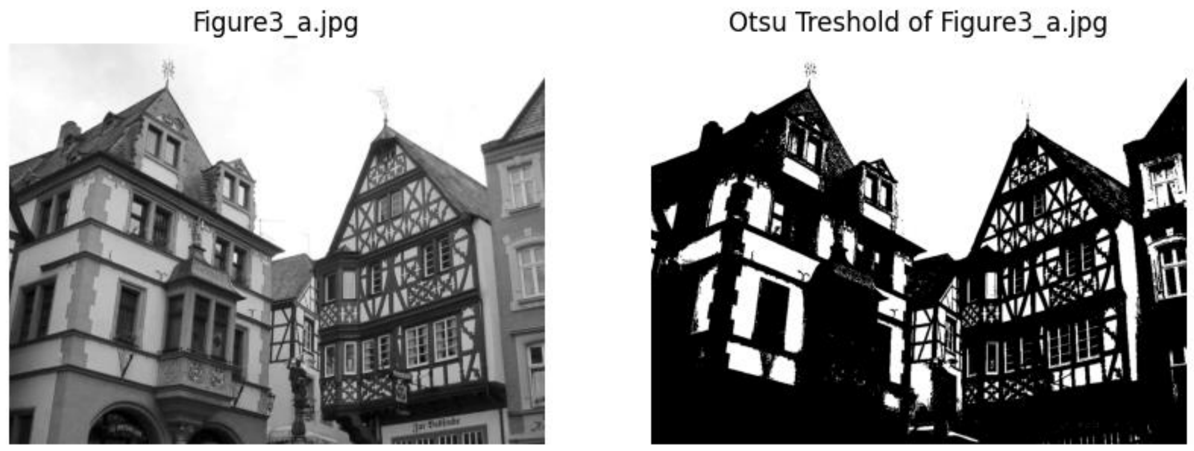


Figure 3.1: Generated binary image with otsu tresholding for figure3_a.jpg

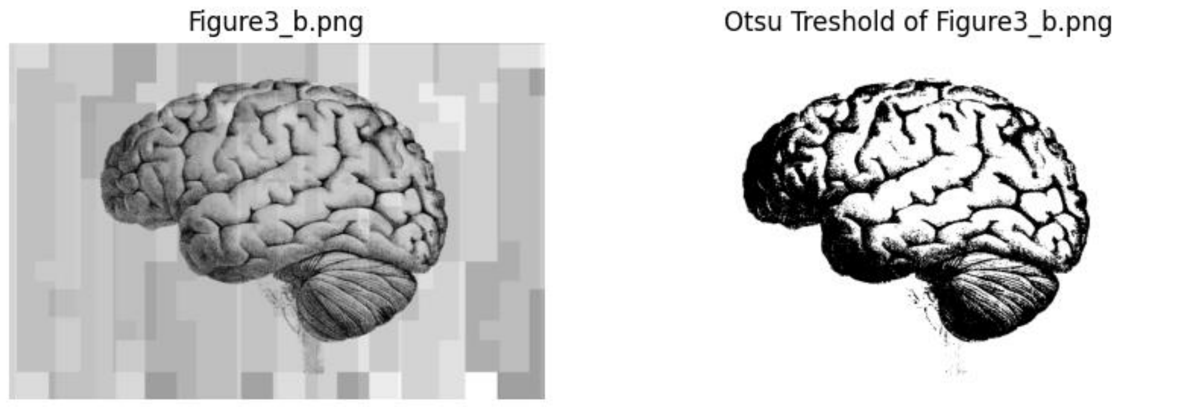


Figure 3.2: Generated binary image with otsu tresholding for figure3_b.png

In terms of results, it can be seen that they are quite good in terms of seperating the background from the foreground. However, they are not perfect.

In Figure 3.1, it can be seen that the some of the higher details are lost as a complete blacked out segment. For example, the middle part of the left building has lost some details due to otsu tresholding. This is because although it looks at in class invariances, otsu tresholding does not take into account how these pixel values are displayed with respect to each other. In other words, the distribution of the pixels with respect to each other are not taken into in otsu tresholding. Due to this, some parts of an image can be fully blacked out as a result of being a darker region.

In Figure 3.2, it can be seen that the leftmost side of the brain presented in the image is shaded into complete darkness which resulted in some lost details. This is due to uneven illumination on the image. Although the brain is the foreground, as the lightning on the brain is not evenly distributed, the leftmost side is classified as background.

Q4)
In order to implement convolution method, the following third party libraries have been used:

- **Numpy:** for converting the image to the matrix and applying faster operations on the matrix.
- **Matplotlib.pyplot:** in order to display the histogram.
- **PIL.Image:** in order to read the image.

The implemented method takes a grayscale image matrix and is feasible for any grayscale image and filter.
The used filters for the convolution operations are the following:

Sobel horizontal:

-1	0	1
-2	0	2
-1	0	1

Sobel vertical:

-1	-2	-1
0	0	0
1	2	1

Prewitt horizontal:

-1	0	1
-1	0	1
-1	0	1

Prewitt vertical:

-1	-1	-1
0	0	0
1	1	1

After applying the following filters with the convolution operation, the results are the followings:

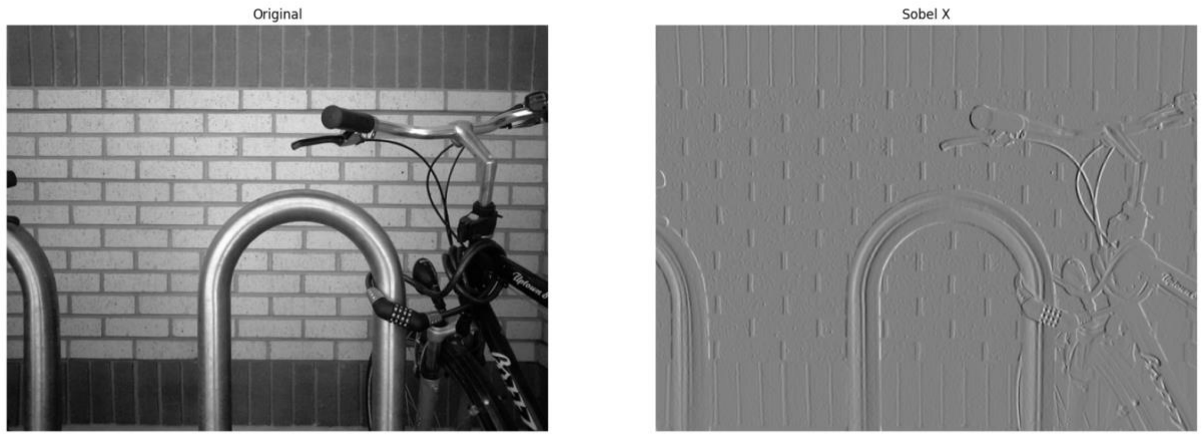


Figure 4.1: Convolution operation on Figure4.jpg with sobel horizontal filter

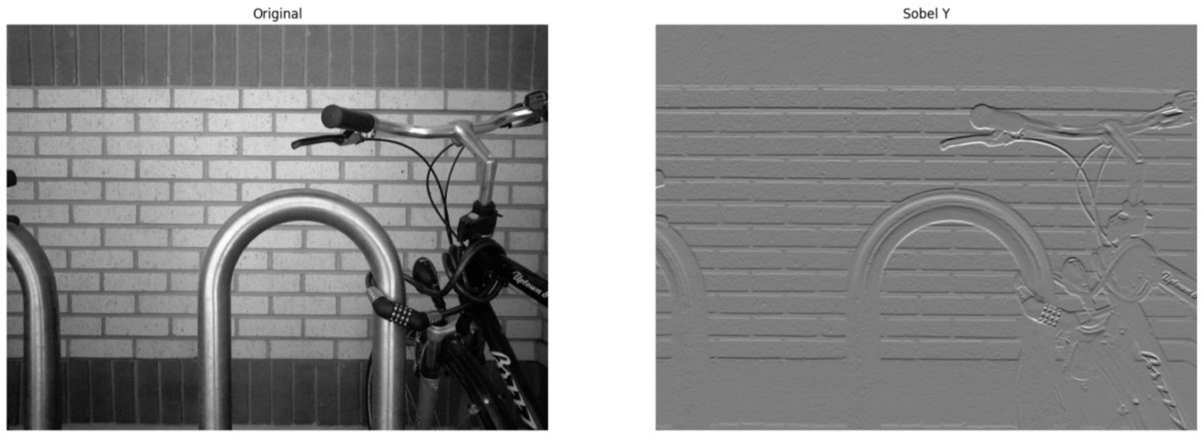


Figure 4.2: Convolution operation on Figure4.jpg with sobel vertical filter

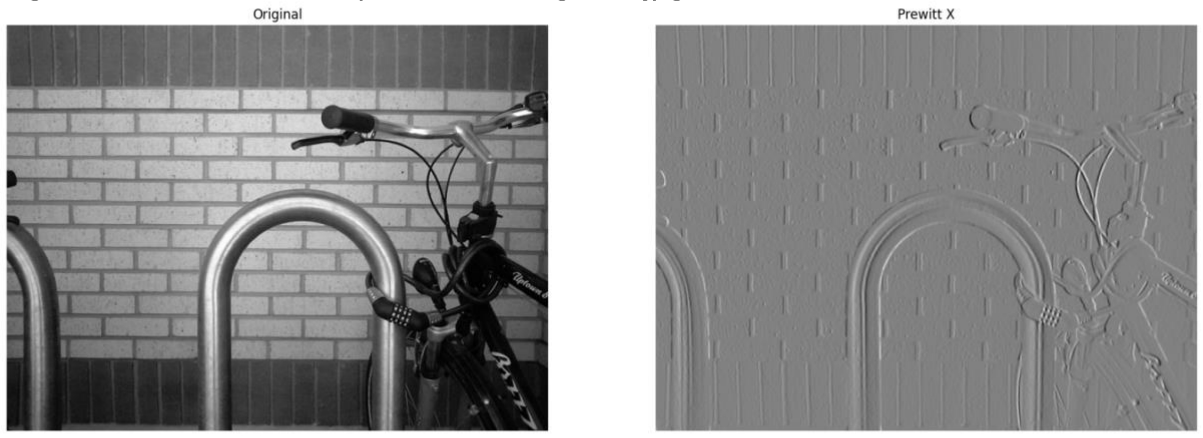


Figure 4.3: Convolution operation on Figure4.jpg with prewitt horizontal filter

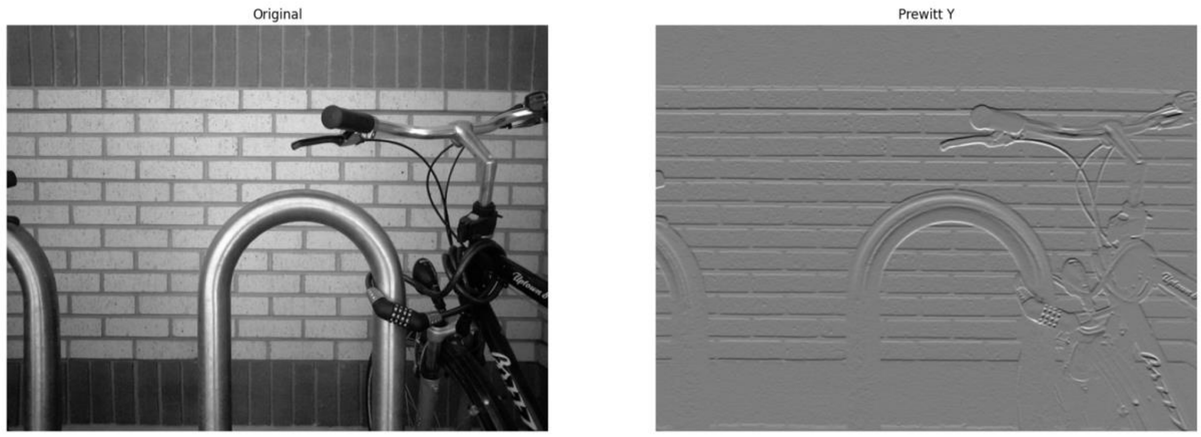


Figure 4.4 Convolution operation on Figure4.jpg with prewitt vertical filter

As it can be seen, horizontal filters highlight edges with the horizontal directions while vertical filters highlight edges with vertical directions. This is because of the filter shape. By nature, horizontal filters look at the horizontal gradients (dx) by subtracting the right side from the left side for each pixel. On the other hand, vertical filters look at the vertical gradients (dy) by subtracting the top from the bottom.

This effects is visually clear in the wall of the filters. The wall in the background contains both vertical and horizontal edges, but the filters nature decides which direction to take into account.

Also, as sobel operators contain 2 rather than 1 in the middle of the gradients, their edges are more intense. This is because the gradient is multiplied by 2 while calculating which results in sharper edges in the filtered output.