CS 484 Spring 2022 Homework 1

Full Name: Zübeyir Bodur **ID**: 21702382

1. Noisy Car Plate

For this question, the following structuring element is used:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The reason for using 0's instead of 1's is that, for the given binary image, we have black (zeros) as the foreground to be dilated or eroded, and foreground as white. Let the given car plate image be A, and structuring element B. After performing the following, we get M_1:

$$M_1 = (A \ominus B) \circ B) \cdot B$$

$\cdot$ *stands for closing,* $\circ$ *stands for opening,* $\oplus$ *stands for dilation,*

$\ominus$ *stands for erosion*

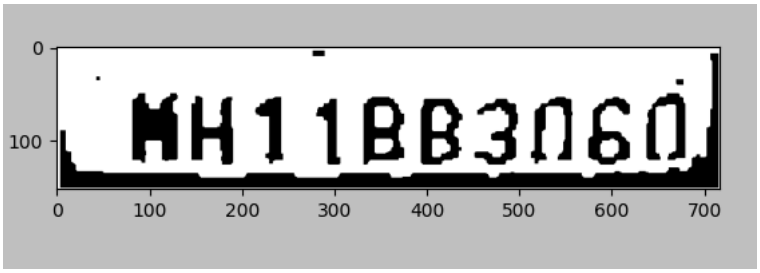

Figure 1: Matplotlib image of M_1

Then, we can further modify this image to come up with the following:

$$M_2 = (M_1 \ominus B) \circ B$$

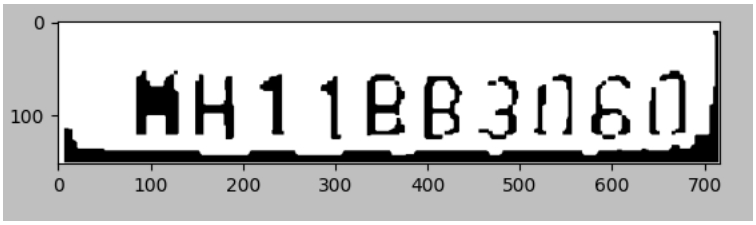

Figure 2: Matplotlib image of M_2

In Figure 1, we can observe that most of the pepper noise is removed, and characters are legible. For the remaining two noisy pixels in Figure 1, I tried the morphological operations to get M_2, however this resulted characters to have gaps in them. So, I tried the following to get the final result:

$$M_3 = (M_2 \oplus B) \oplus B$$

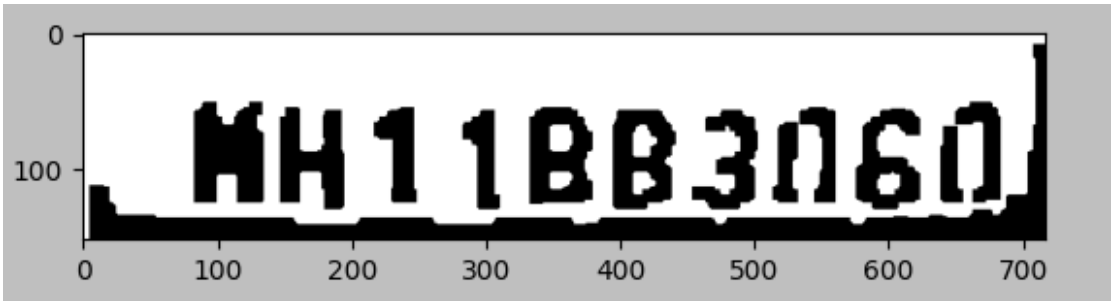

Figure 3: Matplotlib image of M_3

It should be noted that the final result is still not perfect, as there are gaps in 0 characters, and the first character is still not legible. In other words, an OCR algorithm would not be successful to detect the first character as 'H'.

In the implementation, for array manipulation, NumPy, for reading images Pillow, and for displaying results, MatPlotLib, and also math library were used.

2. Histogram Generation for Grayscale Images

For the images "grayscale_1.jpg" and "grayscale_2.jpg", we get the following histograms respectively:

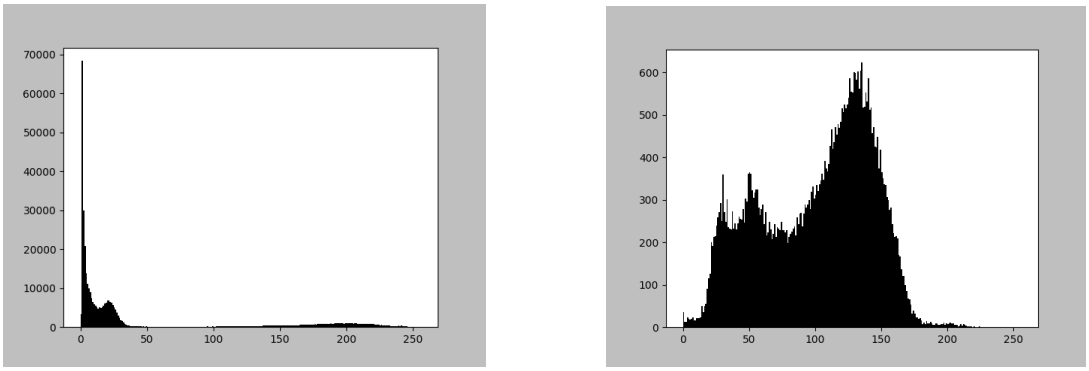

Figure 4: Histogram for "grayscale_1.jpg", and "graycsale_2.jpg".

The implementation uses same libraries as in question 1.

3. Otsu's Method

For the first image, "otsu_1.jpg", the foreground and the background was partially separated. Optimal t value found was 193:

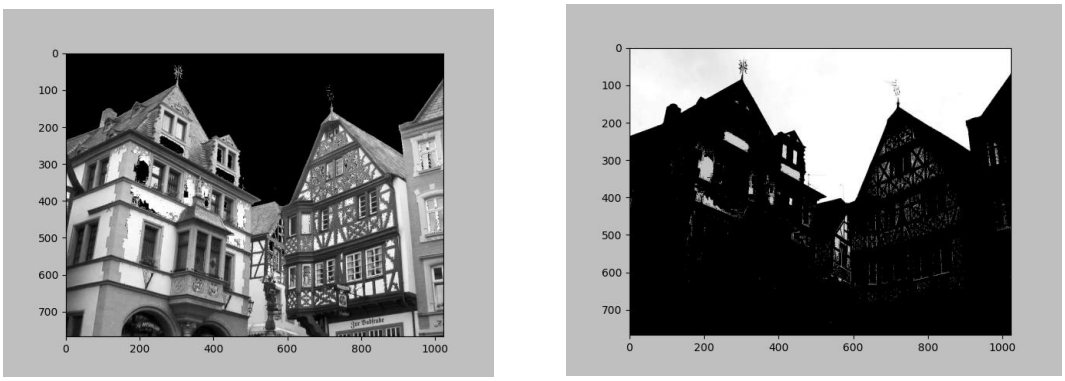

Figure 6: Partitioned foreground and background for "otsu_1.jpg"

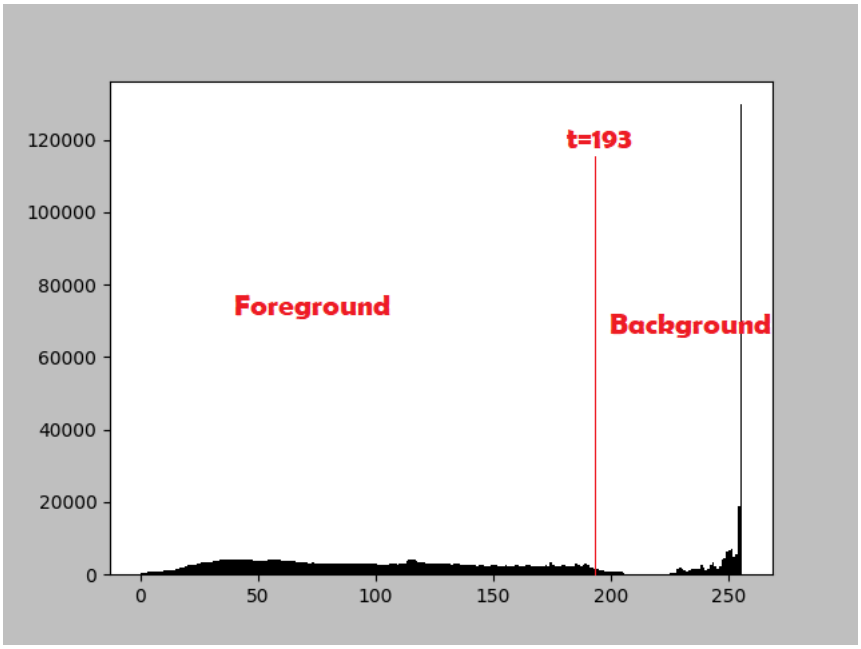

Figure 7: Histogram for "otsu_1.jpg"

From the results, it can be seen that Otsu's thresholding may have small errors in detecting the foreground, as the thresholding does not guarantee that a pixel in one range will always be in the foreground or background.

For the second image, we can observe this flaw in the algorithm very well. Optimal t value found was 82:

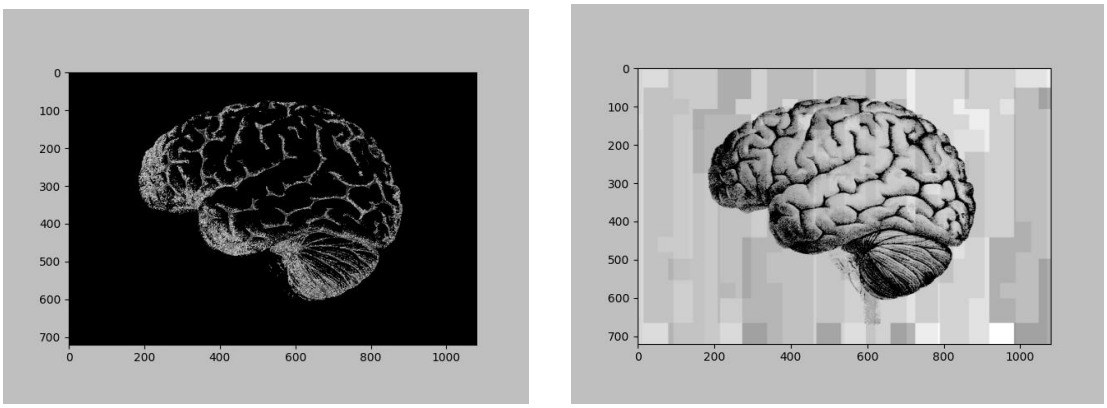

Figure 10: Partitioned foreground and background for "otsu_2.png"

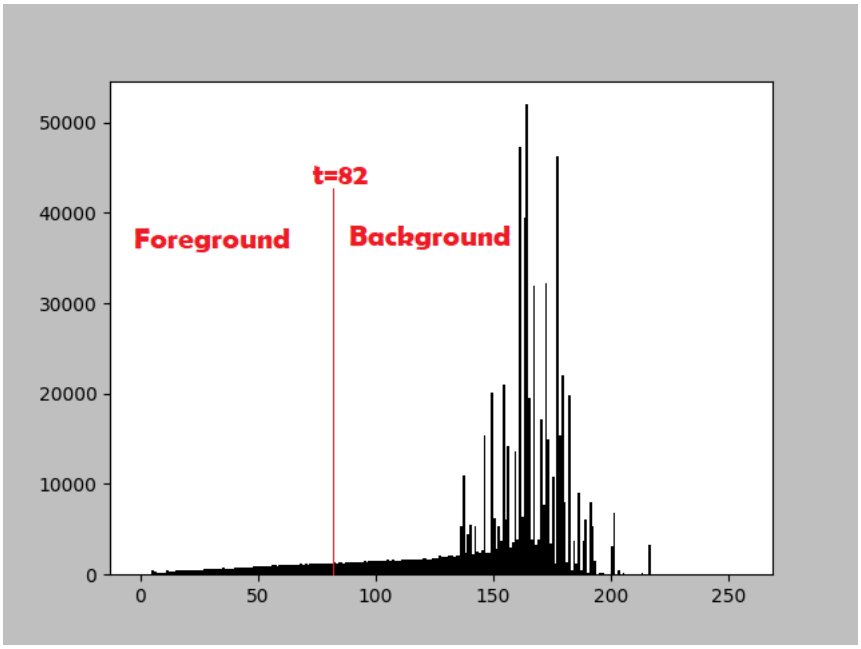

Figure 11: Histogram for "otsu_2.png"

For the second image, the background acts as a noise, because background pixel values are so close to the values in foreground. This confuses the Otsu's Method, as brain figure seems to belong in the background just because it shared similar pixel values with these noises. The way the background has such texture (constant pixel values at large areas) also confuses the histogram, as it dramatically affects the variances of each partition.

Hence, Otsu's method does not always generate perfect results, as a pixel belonging to an interval does not necessarily mean it will only appear in a foreground

or background, and there will be serious problems if the image has noise that disturbs the partitions too much.

Implementation uses same libraries in Q1 and Q2, as well as sys library.

4. Sobel & Prewitt Edge Detection

Below are the outputs for edge detection for Sobel and Prewitt filters. First, df/dx and df/dy gradient vectors for those images were computed. Then, those vectors magnitude was computed and displayed as an image. The outputs have really small differences in quality. Implementation uses same libraries in Q1, Q2 and Q4:

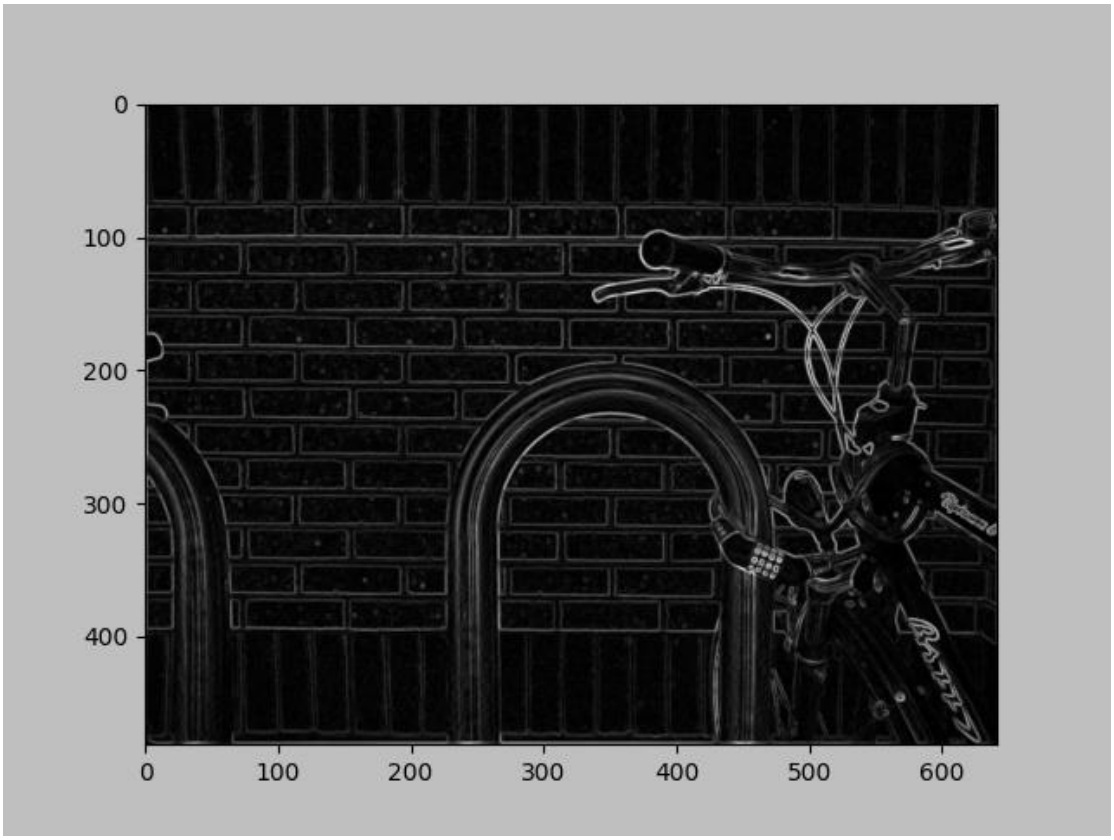

Figure 11: Sobel edge detection output

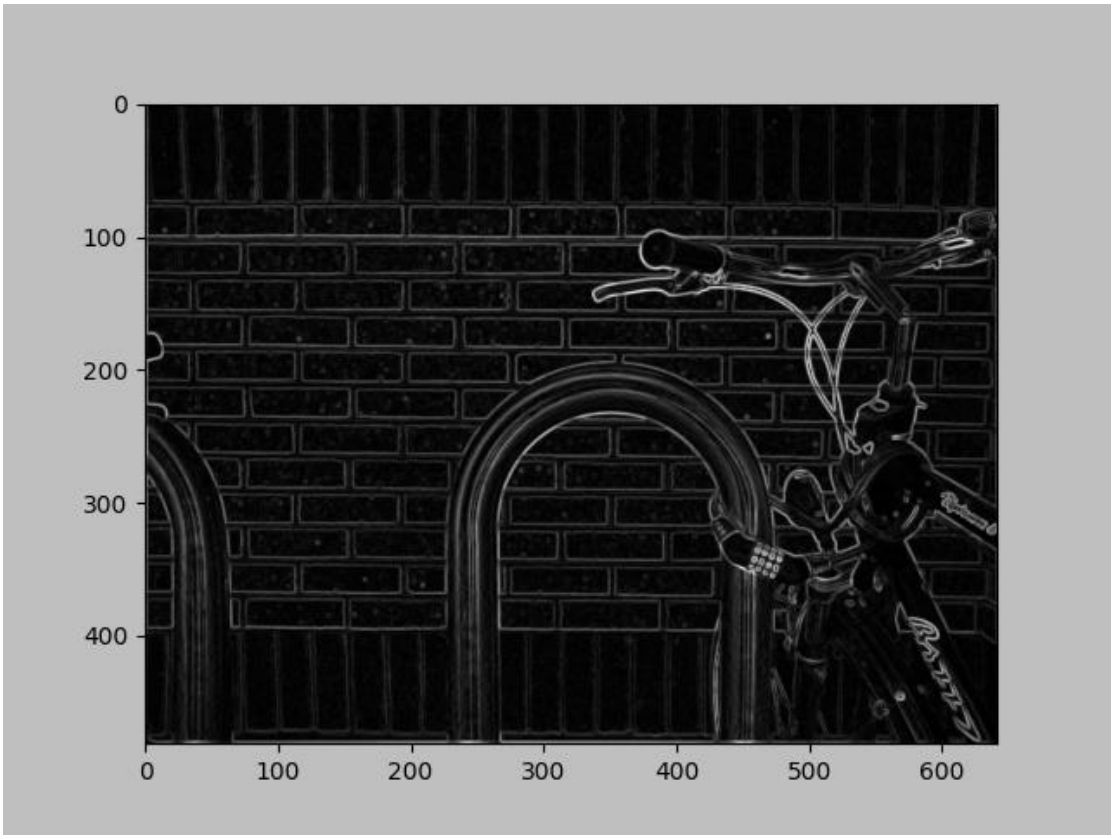

Figure 12: Prewitt edge detection output