CS 484 Introduction to Computer Vision

Homework 1

Ege Türker 21702993 Section 1

Instructor

Shervin R. Arashloo

Question 1

Note: This part was done before the correction mail for the image. However, I had already converted the grayscale image to a binary image in my code so I didn't change it.

The python script for question 1 is q1.py and it includes functions for dilation, erosion operations and an additional function for reading the image.

- **read_img(img_name):** This function takes the name of the image in the script directory and converts it to a grayscale matrix using cv2.imread function. Since erosion and dilation operations work on a binary matrix, cv2.threshold function converts the grayscale image into a binary image matrix with the threshold value 127. The function returns the binary image matrix.
- **dilation(src_img, struct_et):** This function takes an image as a binary matrix and a structuring element as a matrix. Structuring element is generated in the main using element_x and element_y constants. The structuring element I used for the images is a 3x3 matrix filled with 1's. However the function was tested for other structuring elements (5x5, 1x5 matrices and matrices with different shapes) and it works for all kinds of structuring elements. The function puts the center of the structuring element on each pixel of the binary image and compares the structuring element with overlapped pixels. If two 1 values overlap, the corresponding pixel on the result image matrix is set to 1, otherwise, it's 0. Out of bounds pixels are considered 0. Once all pixels are covered, the function returns the resulting image.
- **erosion(src_img, struct_et):** After taking the image and the structuring element as a matrix in the same way as dilation function, this function puts the center of the structuring element on each pixel of the binary image and compares the structuring element with overlapped pixels. If all of the structuring element matches the area in the image, the corresponding pixel on the result image matrix is set to 1, otherwise, it's 0. Out of bounds pixels are considered 0. Once all pixels are covered, the function returns the resulting image.

The results of the operations are given below. Original name plate image is blurry and noisy. After the dilation operation, the noise was removed, however the characters on the plate are now thicker because of the extra black pixels added. Similarly, after the erosion operation, the noise was removed but it also removed too many black pixels from the characters. In order to get a clean image, I applied dilation first, and then erosion to the dilated image. Dilation removed the noise and erosion removed the unnecessary pixels added by dilation. Resulting image was close to the original image with the noise removed.

Original image "name_plate.png"

Image after dilation “dilated.png”

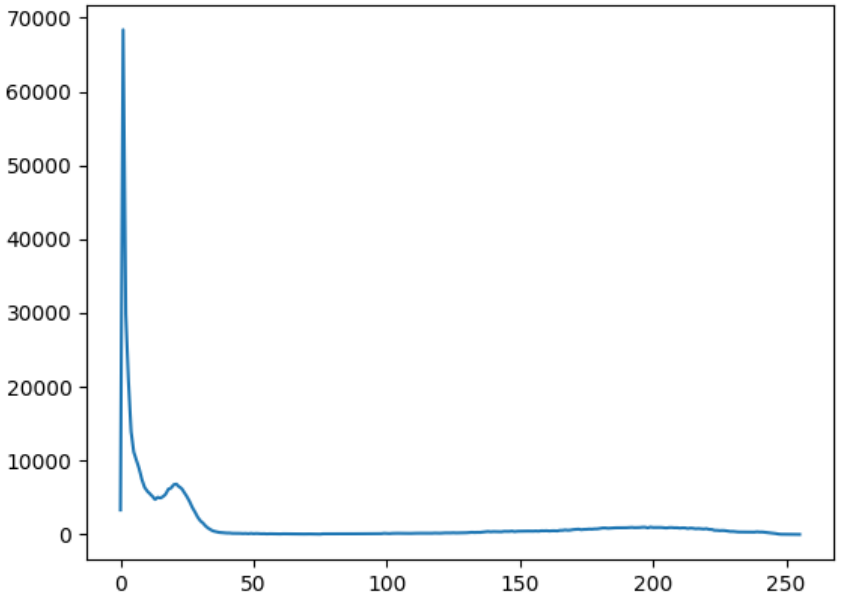
Image after erosion “eroded.png”

Image after sequentially applying dilation and then erosion “final.png”

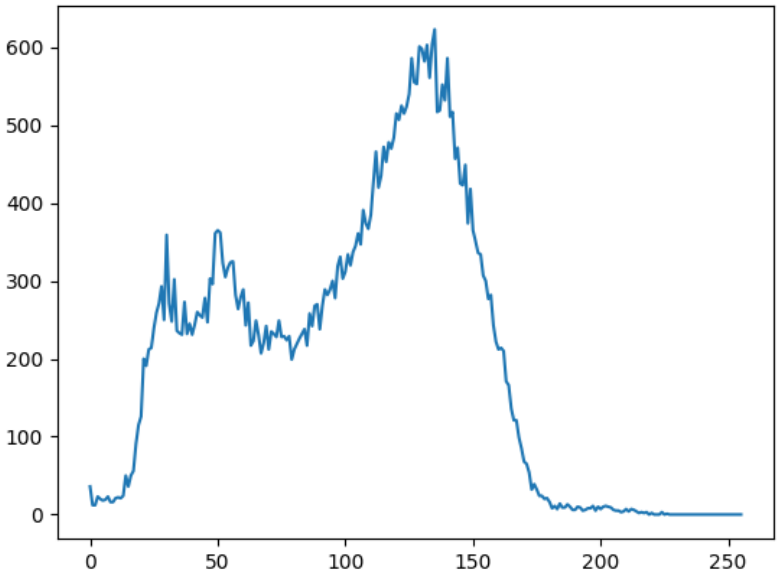
Question 2

The python script for question 2 is q2.py and it includes the function histogram. This function creates a list of size 256. Each index represents the grayscale values from 0 to 255. Then, the function goes over all of the image matrix, counts the number of pixels for each color and increments the corresponding index of the list.

Main is reading the grayscale image, calling the described histogram function and then puts the returned values in a graph using matplotlib library.



Histogram for grayscale_1.png. Most of the pixels are accumulated in the lower grayscale values, since there are lots of dark areas in the image.



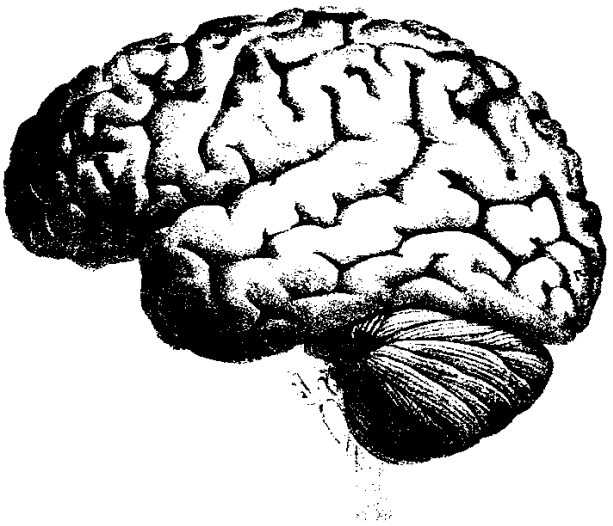
Histogram for grayscale_2.png. This image is more balanced than the first one. Looks like a normal distribution except the 25-75 area.

Question 3

The python script for question 3 is q3.py and it includes functions otsu_threshold and remove_bg.

- **otsu_threshold(img):** This function calculates the threshold using Otsu's method. It first flattens the two dimensional matrix into a one dimensional array. Then from the minimum grayscale value the image includes to the maximum grayscale value, it calculates the background and foreground variance values for each grayscale value. Using the background and foreground variances, it then calculates the in class variance. In the end it returns the threshold value where the minimum in class variance. Essentially, it calculates the ideal threshold value for background removal.
- **remove_bg(img, threshold):** This function sets the pixels under the given threshold values to 0 and pixels over the threshold value to 1. Separating background with foreground using the threshold value calculated in the first function.

Result for the image "otsu_1.jpg". For the sky, the method works perfectly. But since some parts of the houses are light colored, they are also treated like background and are set to 0.



Result for the image “otsu_2.png”. Background is fully removed. But similar to the first image, parts of the brain are also set to 0.

Question 4

The python script for question 4 is q4.py and it includes the function convolution. This function takes an image and a filter and applies the convolution operation on the image. It centers the filter on each pixel of the image (except the bounds), and a matrix multiplication on the area and the pixel. Then takes the average sum of the matrix and puts it in the centered pixel. This function works for all types of filters.

The main method uses Sobel and Prewitt filters. After the image is read, the convolution method is applied horizontally and vertically on the image. Then the two results are added to each other.

Sobel filter applied on “filter.jpg”, highlighting the edges in the image.

Prewitt filter applied on “filter.jpg”, highlighting the edges in the image.