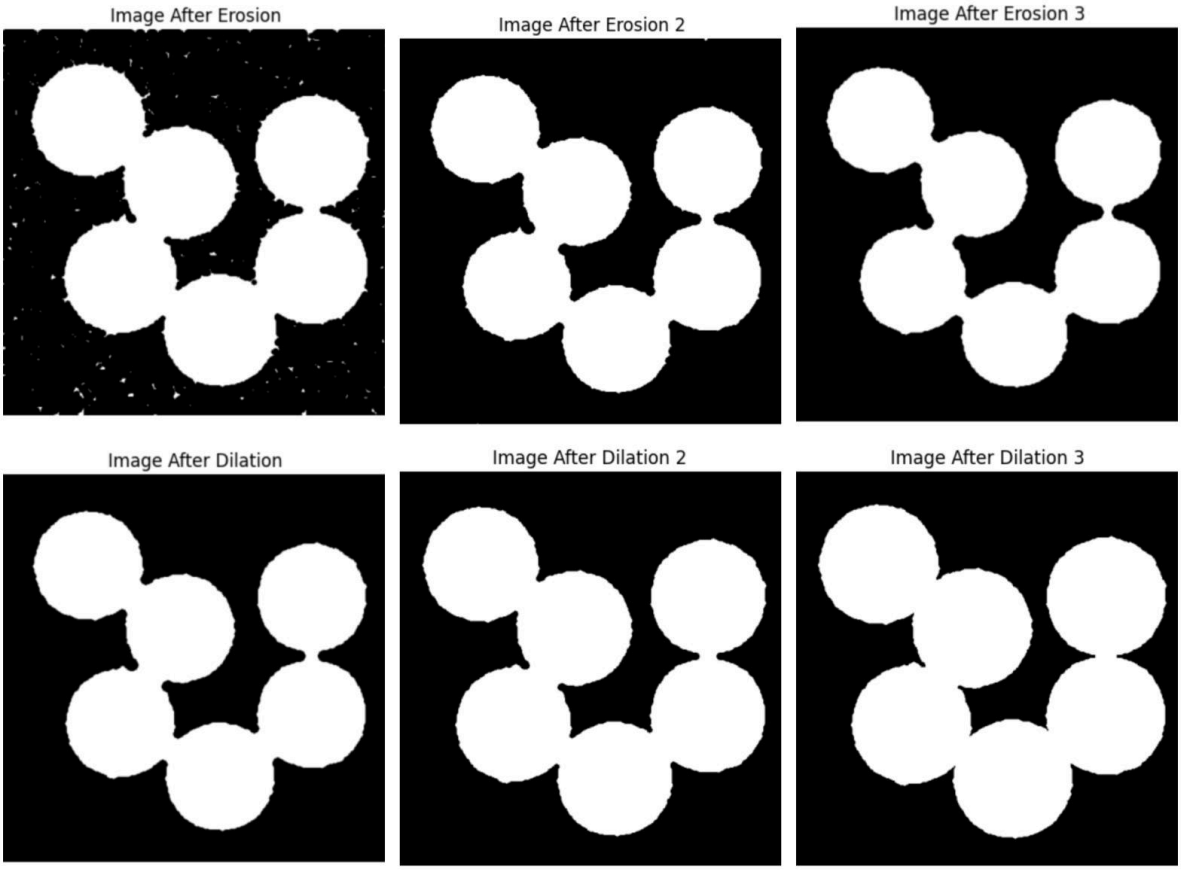
CS484/555: Introduction to Computer Vision Fall 2024
Homework Assignment 1

Due: October 28, 2024 Yasemin Akın 22101782

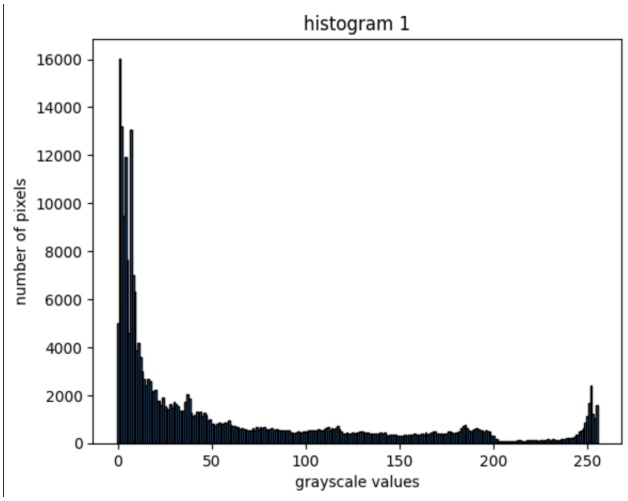
Morphological Operations

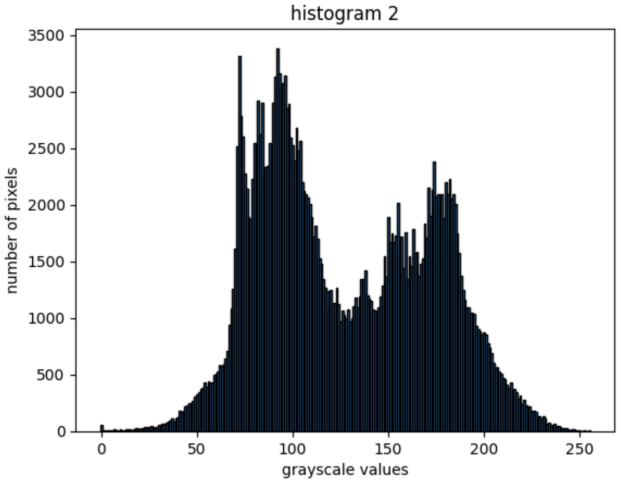


Applied 3 back-to-back erosion operations to be able to fully get rid of small holes at the background. After the second erosion operation, the small circles at the back seem to have closed, but in reality that is not the case, there are still small white dots above and below. When I performed a dilation operation on this version of the image, I saw that those white circles had grown again and became more visible. Then applied 3 back-to-back dilation operations to be able to transform the white circles to their original size. Here, the reason why I have applied 3 dilation operations is to cancel out each of the previous 3 erosion operations and to restore the size of the circles to a size as much as closer to their sizes in the original image.

Histogram-Based Image Enhancement

Part 1

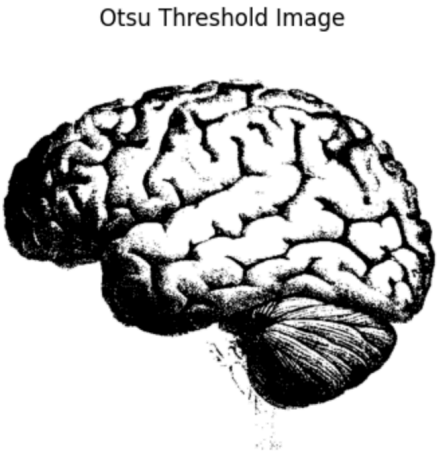


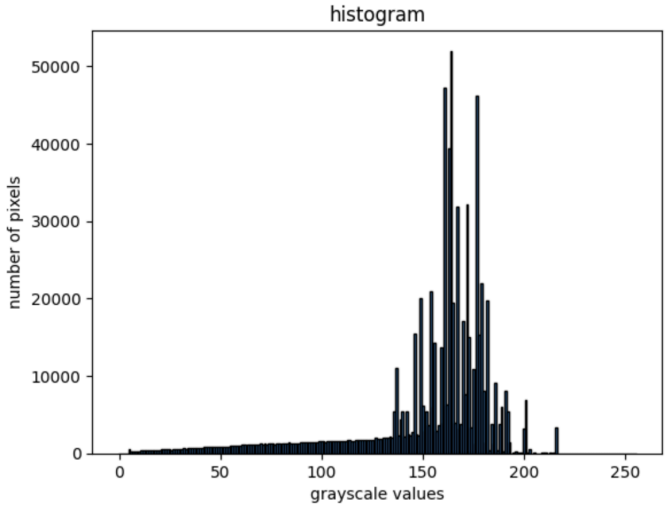


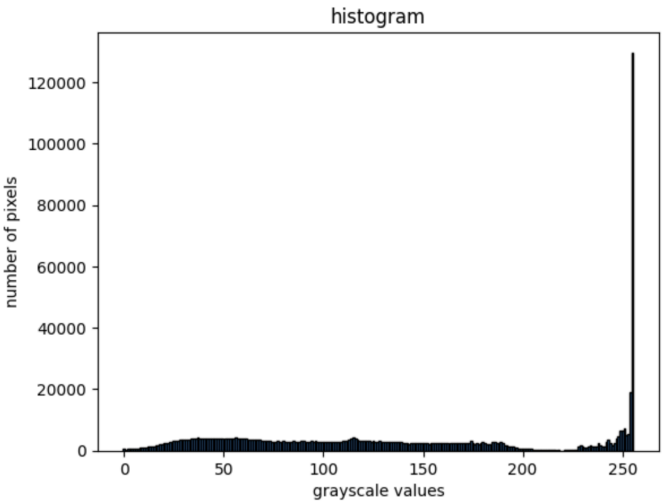
Part 2

- c=0, d=255: Enhanced the contrast by remapping the values to the whole grayscale spectrum. Beforehand, the intensity values were closer to each other, now they are spread; bright regions are brighter, dark regions are darker.
- c=128, d=255: Image appears washed out. Image is brighter since the darkest intensity became 128, meaning darker intensities near 0 are now mapped near 128.
- c=0, d=128: Image appears darker. Brightest parts of the image are now scaled down near 128, appearing as shades of gray, while dark regions remain dark.

Otsu Thresholding







- Threshold found for the brain image = 216
- Threshold found for the building image = 255

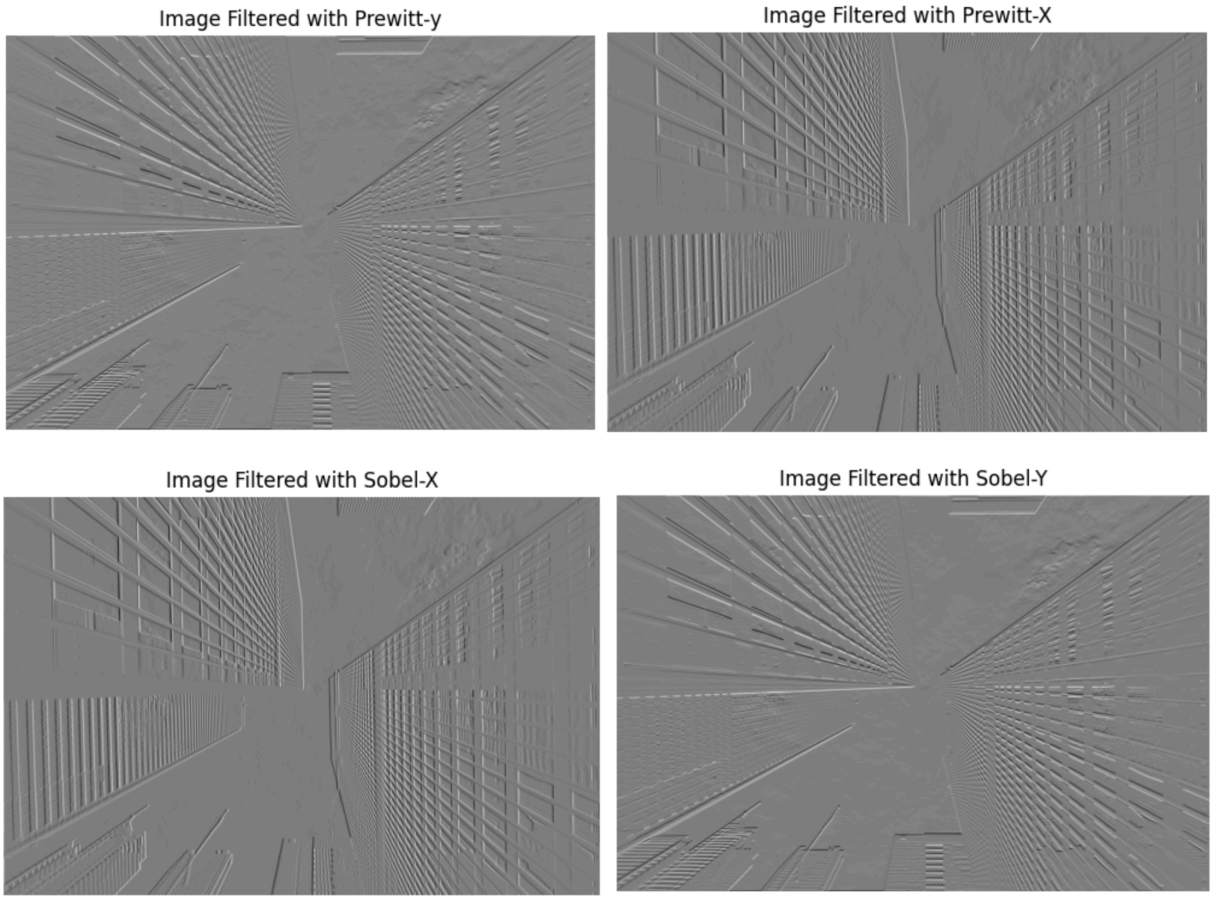
Otsu Thresholding can be thought to be perfect when:

- The thresholded image is completely separated as foreground and background with no parts of each side mistakenly included in the other side.
- Sharp edges in the image are clearly defined.
- There are no remaining background artifacts.
- There is no loss of foreground details.
- The source image is bimodal.

Here, as we can also see from the histograms of the images, the images are not bimodal, meaning inter-class-variances are not high as they should be, as most of the realistic examples of images. Therefore, Otsu Thresholding is not completely perfect in most cases, because of its rare assumption. In our cases, the background and foreground is nearly separated fully, but some noise can be seen in the background of images. Also, a lot of foreground details are lost after thresholding the images, this is also a negative impact of non-bimodality.

2-D Convolution in Spatial and Frequency Domain

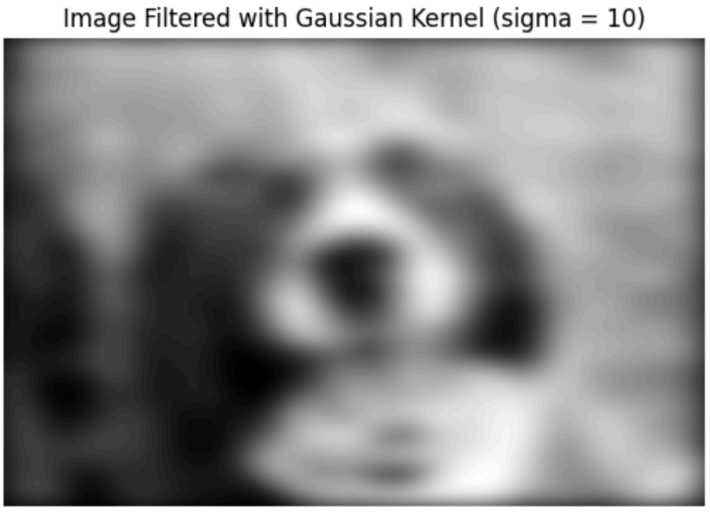
Part 1



- Prewitt X and Sobel X: Emphasized vertical edges by computing gradient (first order derivative) in the horizontal direction. Large rate of changes in the x direction resulted in strong responses. Additionally to the Sobel kernel, it is more robust to noise than Prewitt because of the inclusion of larger weighting elements.

- Prewitt Y and Prewitt X: Emphasized horizontal edges by computing gradient (first order derivative) in the vertical direction. Large rate of changes in the x direction resulted in strong responses. Additionally to the Sobel kernel, it is more robust to noise than Prewitt because of the inclusion of larger weighting elements. It is more sensitive to fine details.

Part 2



To apply the Convolution Theorem to filter the image, first I have followed below steps:

1. Determining the dimensions of a given source image.
2. Pre-processing the image, multiplying it with (-1)^(x+y) to center the transform.
3. Discrete Fourier Transform (DFT) applied to the 2D image matrix.
4. The Gaussian lowpass kernel is created in the frequency domain with sigma=10 and the size of the kernel is equal to the size of the image.
5. Kernel and the Fourier Transformed image are multiplied.
6. Inverse DFT applied to the resulting 2D matrix from the multiplication operation.
7. Post-processing the image, multiplying it with (-1)^(x+y) again. Returning this.

But, in this way, the process was taking too long since the time complexity turned out as O(H^2M^2) and I was not able to get a filtered result because the given source image's resolution is 183x275, which is not low. I have tried to downsample the image to 64x64, that way I was able to get a result, but it still took a long time. Therefore, I switched my approach after doing some research and found out about Fast Fourier Transform (FFT)[1] and Cooley-Tukey[2] implementation of FFT. Time complexity went down to O(HMlog(HM)) and I was able to get the above result in a short amount of time with sigma=10. The process of the FFT that I have applied is like this:

1. Determining the dimensions of the given source image.
2. Centering the image, multiplying it with (-1)^(x+y).
3. Computing the 2D FFT of the image:
    a. Applying 1D FFT to all the rows of the image.
        i. Here, the DFT process is recursively broken down to smaller pieces by the Cooley-Tukey algorithm. A size N 1D matrix DFT is broken down to sizes of two N/2 1D matrix DFT problems. Here also, if N is not a power of two, we pad it to be the nearest power of two, because the Cooley-Tukey algorithm works like this.
    b. Taking the transpose of the image to apply 1D FFT to the columns of the images and then taking the transpose of the image again to convert columns to rows again.

[1] https://en.wikipedia.org/wiki/Fast_Fourier_transform
[2] https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

4. Generating the Gaussian kernel in the frequency domain. The kernel has the same size with the image and sigma=10. The size of the kernel matches the image's because frequency domain filtering requires element wise multiplication of the image and the kernel and instead of adding pads to the kernel I thought that directly generating the kernel to match to the image size is more feasible.
5. Multiplying the kernel with the image, element wise, in the frequency domain.
6. Applying Inverse FFT (IFFT) the same way I am implementing the FFT to the masked image. Using the 1D FFT function to implement the 1D IFFT because IFFT can be calculated by conjugating the 1D matrix and applying direct FFT to the conjugating matrix and then conjugating the matrix back.
7. Post-processing the image, multiplying it with (-1)^(x+y) again to re-center it. Returning this as the output image.

External Libraries Used

- The opencv-python package is installed to use the cv2 library for reading images as matrices in specified formats i.e. grayscale.
- The drive library from the google.collab package is used to mount my google drive to the notebook environment, because I uploaded the images to the drive and reached them from there.
- The matplotlib.pyplot package is used for plotting the images and the plots.
- The math and cmath packages are used for implementing the mathematical expressions in the last question, which is the frequency domain filtering with Fourier Transform question.