Bilkent University

CS484 – Introduction to Computer Vision

Fall 2024 - 2025

Homework 1

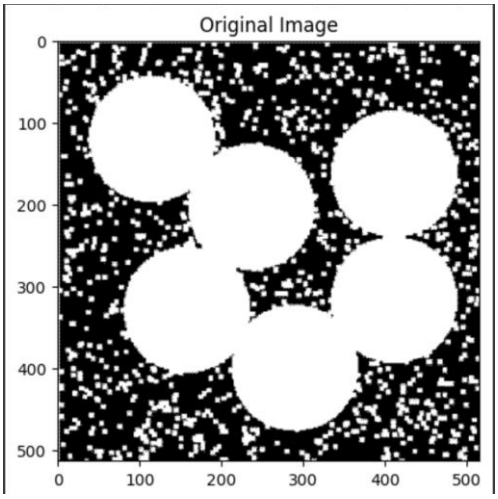
Emre Melih Güven

22003944

Computer Science

Question 1 – Morphological Operations

First, before applying dilation and erosion, the image is converted to grayscale by using NumPy library. Secondly, by setting 128 as a threshold pixel intensity, I obtained the binary image.



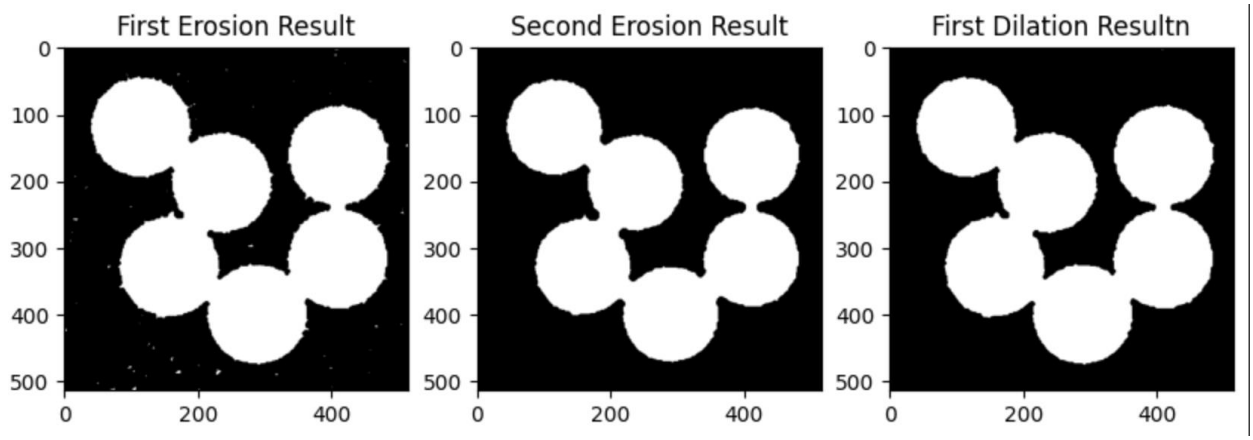
Later, as a structuring element, I used 7x7 array. 7 is chosen due to the dense white noise in the background. First, I tried to apply 3x3 and 5x5 but these filters were not sufficient. To use 3x3 or 5x5, I had to apply erosion more than twice and dilation more than once. To decrease the steps and computation load, I used 7x7. The structuring element is shaped as a circle since the objects in the photograph are circular and to maintain the round shape better, I didn't use np.ones directly and made a custom structuring element.

```
# 7x7 due to the circular shape and high noise ratio
structuring_element = np.array([[0, 0, 1, 1, 1, 0, 0],
                                [0, 1, 1, 1, 1, 1, 0],
                                [1, 1, 1, 1, 1, 1, 1],
                                [1, 1, 1, 1, 1, 1, 1],
                                [1, 1, 1, 1, 1, 1, 1],
                                [0, 1, 1, 1, 1, 1, 0],
                                [0, 0, 1, 1, 1, 0, 0]], dtype=np.uint8)
```

After obtaining the relevant sources, I applied the dilation and erosion algorithms. On dilation, I used padding to obtain total, correct results. Padding is obtained by using //2 operator which makes it flexible under different structuring elements. Padding value is decided as 0 so that padding won't affect if any pixel is 1. So, padding won't corrupt the process. Using the padded array, I applied structuring element and if there is any 1 in the frame, that pixel is recorded as 1, saved to dilated array which is initialized as zeros.

On the erosion, again I applied padding. This time, to not corrupt the process, padding values are initialized as 1 since all pixel values should be 1 in the frame during erosion process. After, I calculated if all values are 1 in the frame using the structuring element, eroded array value is changed to 1. Eroded array is complete zeros initially.

As a sequential morphological operation, I applied erosion once to observe if the noise is removed from the background. However, it wasn't sufficient. Thus, I applied erosion twice. As a result, I removed the noise from the background totally but circles in the image were smaller. Thus, I applied dilation once after erosion to have same circle size.

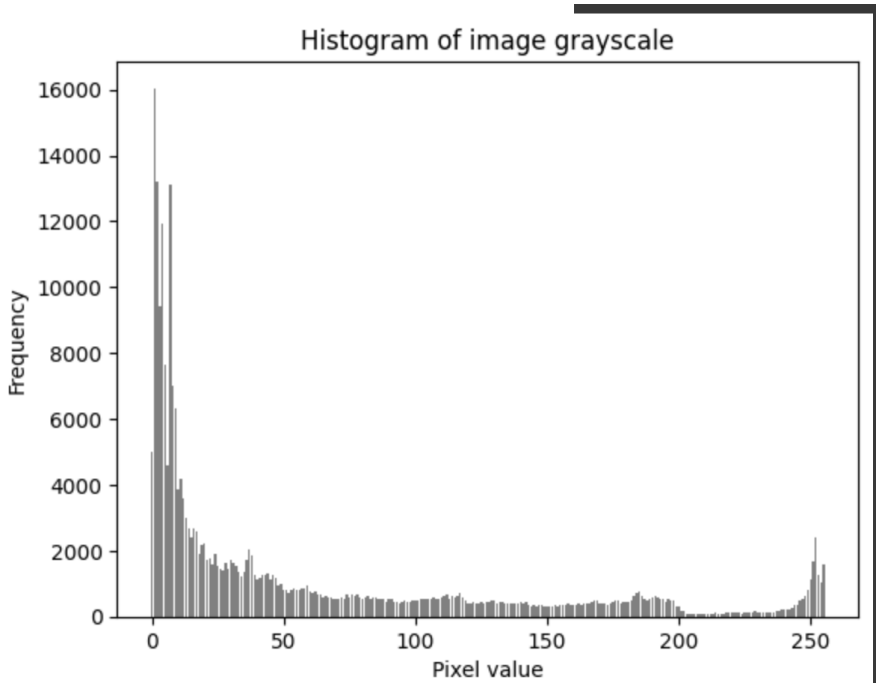


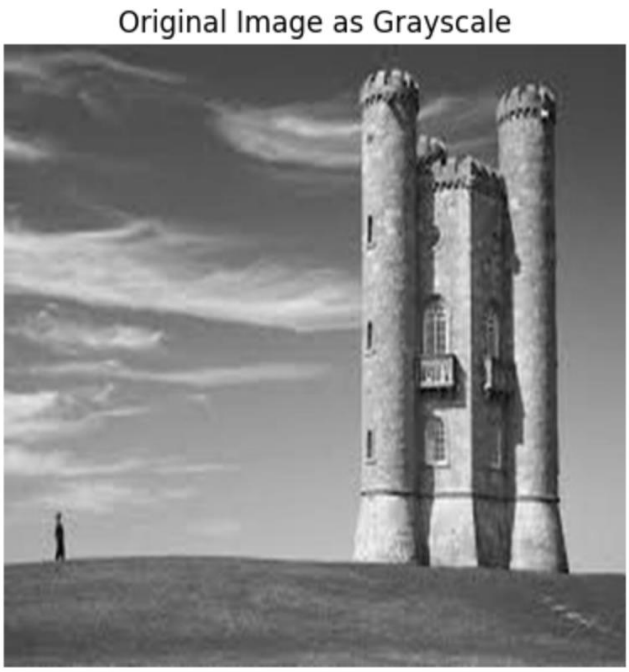
Question 2, Part 1 – Histogram

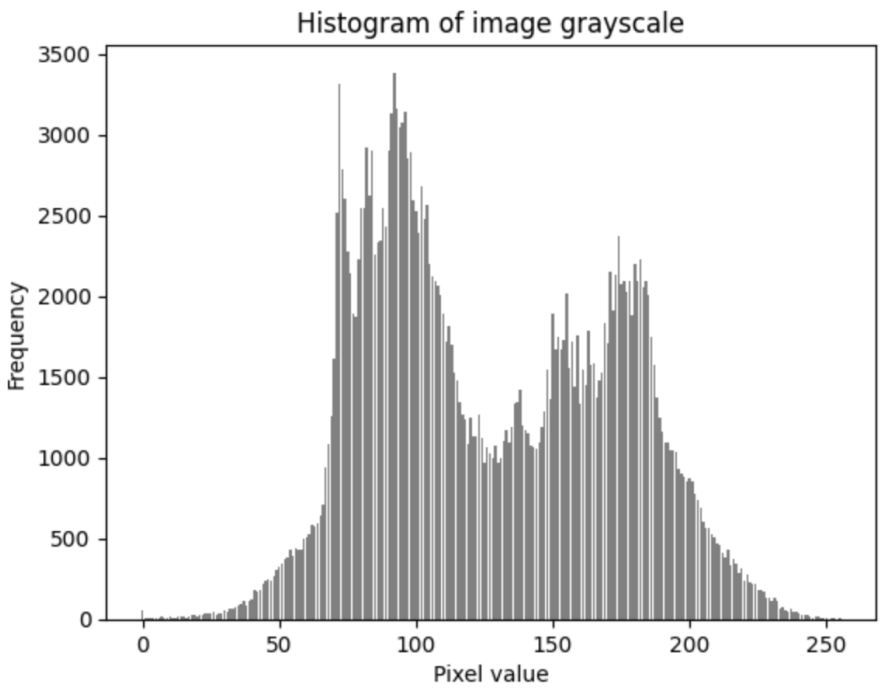
First, I started with converting the image as binary. Since in the first example I used manual thresholding to obtain binary image. This time I used Colab's files library to obtain the wanted image. Using NumPy, I converted the image to np array.

To start histogram, I created an array initialized as zero values that has size 256. Using nested for loops, calculated the occurrence counts of each pixel and recorded them to histogram array. Using MathPlotLib's bar function in range of 256, I displayed the histograms.


Original Image as Grayscale


Histogram of image grayscale
Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
Pixel value





Question 2, Part 2 – Contrast Stretching

First, I took the image as input and converted it to grayscale. Using grayscale, I obtained the NumPy array. Applied the contrast stretching algorithm using source image and c, d values and took the c, d values from an array of tuples. First, I started with calculating maximum and minimum pixel intensity values using NumPy array values and its min and max function.

I applied the contrast stretching algorithm using original image's difference from the min divided by difference between max and min. Than I multiplied the result with d – c which are the parameters and summed c. As a result, I obtained three different results with different c, d values.

The biggest challenge in this part was related to MathPlotLib. When this library is used, it automatically displays photographs as contrasted. I researched hard and while displaying, I understood that explicitly I need to write vmin and vmax to be in range of 0 to 255. Otherwise, the difference in photos were so small that it could be barely visible with naked eye.

Original image

Here, it is visible that using c as 0 and d as 255 made the pixel intensities in wider range. This resulted in a picture which has a wider range of pixel intensities. Thus, photograph looks more detailed and with balanced white and dark pixel intensities.

Using c as 128 and d as 255, I obtained an image which has higher pixel values. Thus, it is visible that the picture is lighter, varying in higher pixel values. In this example, since it is stretched using 128 – 255, dark details are mapped into upper close area of 128 and this made image lighter and lost dark pixel values.

Using c as 0 and d as 128, I obtained a darker image. Since pixel values which are high are mapped into lower close part of 128, making the higher pixel values that represent lighter areas of original image darker. Thus, the image here is darker and varies in the range 0 – 128.

Question 3 – Otsu Thresholding

I started with obtaining the image as grayscale, then having its array using NumPy for pixel intensity. I continued creating a histogram like in the second question. After that, I calculated the probability of each pixel occurrence and saved these probabilities in same array changing values.

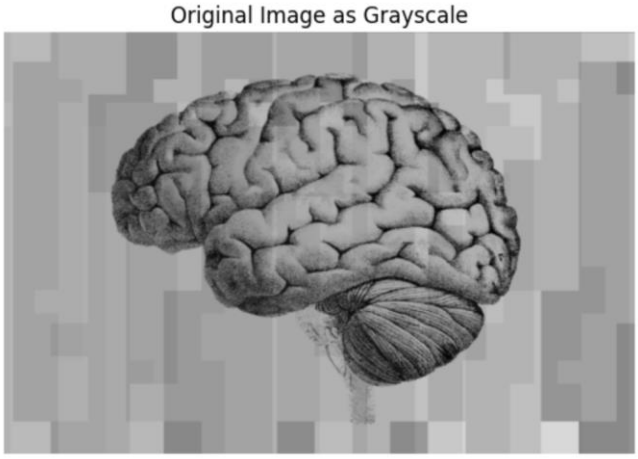
Later, using these probabilities of each pixel's occurrence, I calculated the cdf as a cumulative sum of pdf. Then, using cdf and the pixel value, I obtained the mean value of pixels. Total mean is obtained by the last element of this cumulative mean.

In class notes, I saw that on Otsu Thresholding, the book uses minimal weighted sum of within-class variance. To do so, I started a loop which iterates through each pixel value in range 256. In each iteration, I needed to classify first I pixels as background and rest as foreground since between the spikes, I am trying the optimal threshold of histogram. Using cdf as w_b, calculated w_d using their sum as 1, obtained from cdf.

If there is no separation, the difference between these two regions, passed the calculations to decrease the computation load.

Later, using cumulative means of the separate parts divided by cdfs, obtained mean values. Using each pixel's means of foreground and background, I calculated within class variance using separate variances multiplied by cdfs' summation. Keeping the record of minimum variance by comparing, I obtained the threshold pixel intensity. Then I applied the threshold and obtained the final images.

Image 1:



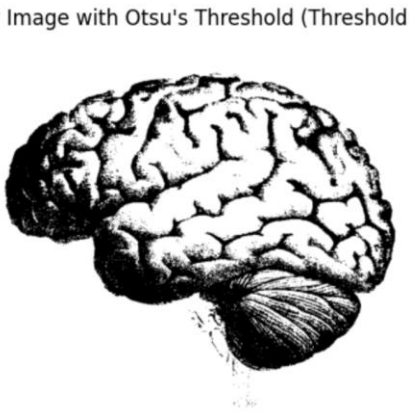
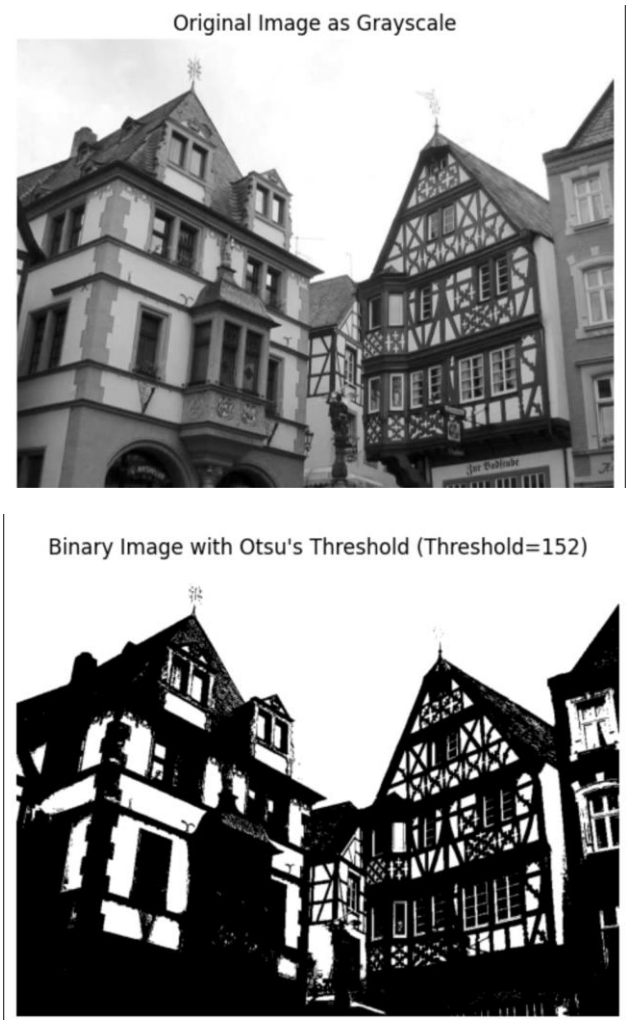


Image 2:



Here, it is visible that the pixel values are threshold in a way that background and foreground is separated from each other successfully. So, bimodal histogram is achieved in the sense of clustering pixel intensities into two distinct groups.

It is not perfect since the histograms of the images are not having two identically same spikes. This is the assumption of Otsu Thresholding. Also, Otsu Thresholding uses global, whole picture, scope of threshold. This causes the confusion of objects and foreground, background.

Since the original images have distinct foreground and background, they resulted as wanted. However, it would not always result in this good.

As a disadvantage, images with more complex pixel intensities, wouldn't result in this good. For example, if images were with poor contrast, intensities would be closer, and it would be harder to differentiate the background and foreground. Also, in noisy objects, they would confuse the histogram and make thresholding more complicated or would need a preprocessing to remove noise.

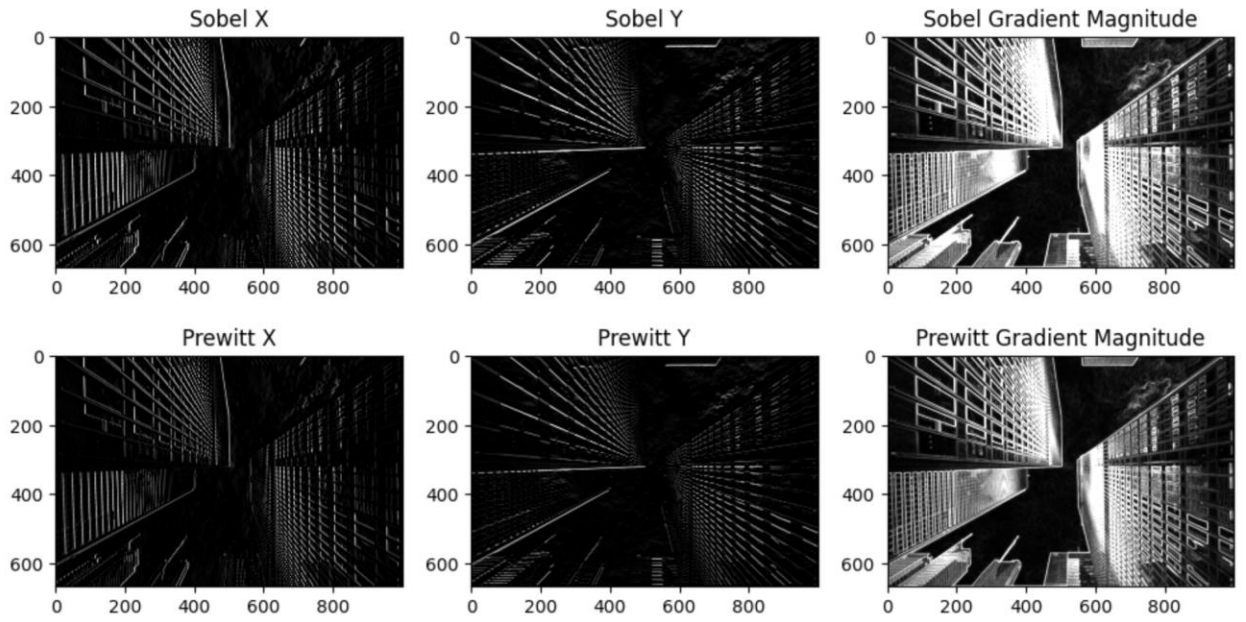
Question 4, Part 1 – Convolution in Spatial Domain

To start with, I took the input image and transformed it to grayscale, then array of NumPy. After that, I created Sobel and Prewitt as 3 x 3 kernels. Using x and y kernels of each, I convolved them with original image.

Convolution is done by first padding the image with zeros. Then, for each pixel, I calculated the corresponding frame and by multiplying and then summing the values of frame and kernel, I obtained the convolved images.

Original image:

Results:



As a result, it is shown that x filters are showing the edges vertically, whereas y filters are showing the horizontal edges. Using these filters' values, I built the gradient magnitude image using Euclidian distance.

The gradients resulted as edges of the images. Sobel is slightly more sensitive to edges due to its weighted center detection. Prewitt doesn't have weight. Thus, Prewitt resulted as thinner edges whereas Sobel displays thicker edges.

The filters focus on the high frequency components of the images which means rapid changes in the pixel intensity values. They smooth the normal areas whereas sharpening the edge transitions.

Question 4, Part 2 – Convolution in Frequency Domain

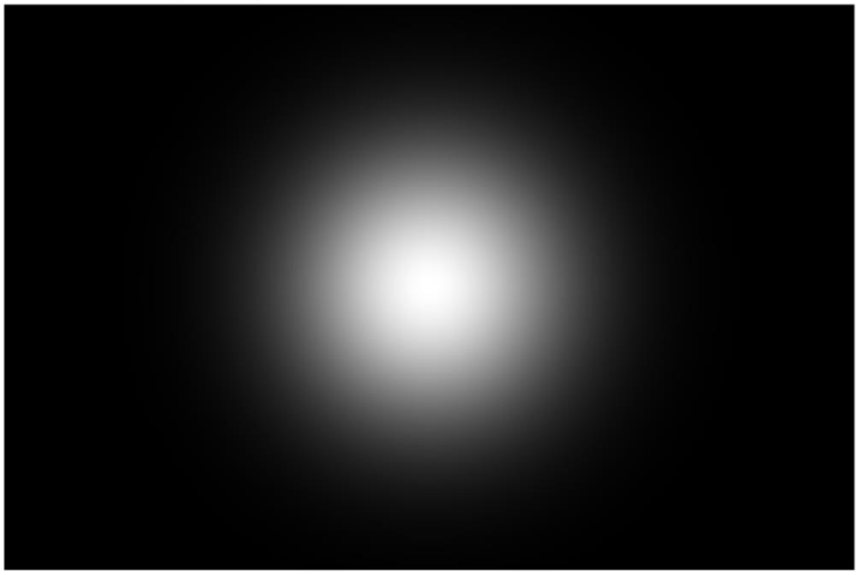
To start with, I took the input image and transformed it to grayscale, then array of NumPy. I defined a gaussian filter function which uses original images shape and sigma. In filter, I centered and using np.ogrid, applied a mesh to calculate distances. As a result of formula, obtained a gaussian filter using sigma and distance calculated overall exponentially.

Later, I applied a frequency filtering algorithm. I started with transforming NumPy array, representing the greyscale values of original, to frequency domain using Fourier Transform. I shifted the result to center the x and y domains to obtain a proper spectrum. Then calling the gaussian filter created with variance equal to 30, I multiplied pixelwise the filter and shifted frequency dimension image.

Using the result, I shifted back the image and using inverse Fourier transform, I got the result. To have a magnitude for real representation, I took the absolute values to make them positive.

Original Grayscale Image

Frequency domain Gaussian Low-Pass Filter



Filtered image using Gaussian Low-Pass Filter

As seen in the result, the original image is blurred using Gaussian Filter since the sigma is chosen large enough.