Division of Electronics and Information Engineering

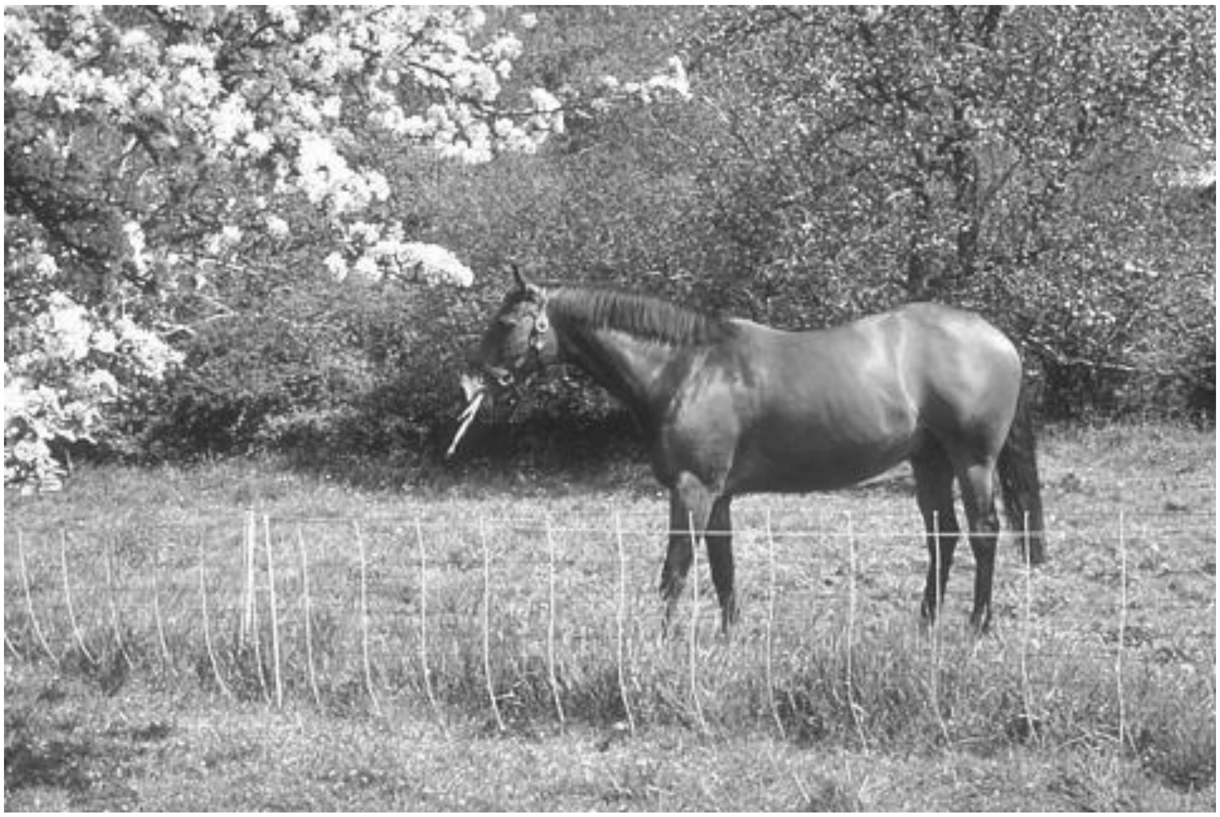
ANALYSIS REPORT PROJECT 2

ADVANCED IMAGE PROCESSING (COMPUTER VISION)

October 3rd, 2020

Reported by: Vuong Minh Phu (ID: 202050798)

Instructor: Prof. Hyojong Lee



(Original Image)

Introduction

In this assignment, I will implement three edges detectors namely: Sobel Edge, Prewitt Edge and Canny Edge. This report includes the results from each detector and analyzes the results, finally is the discussion. To easily visualize the result I will use only the image of a horse to interpret my results.

Sobel Edge Detector

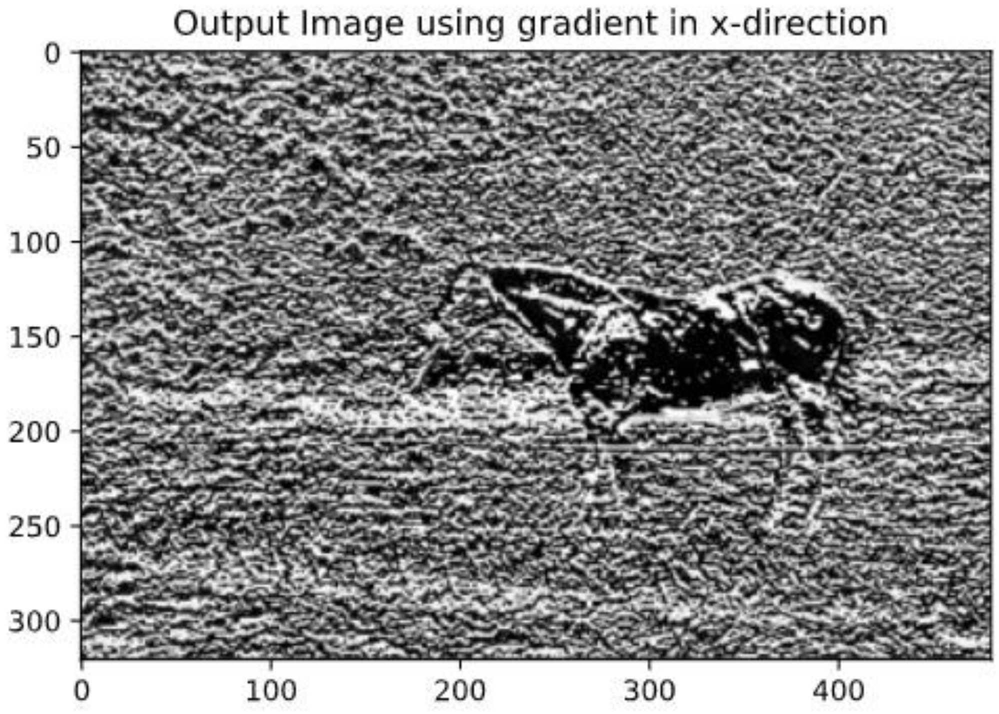
The Sobel detector computes the derivatives in x and y directions, then finds the gradient magnitude and finally threshold the gradient magnitude to generate the edge map.

We can compute the derivatives simply by convolving the image with the derivative matrices in x and y directions.

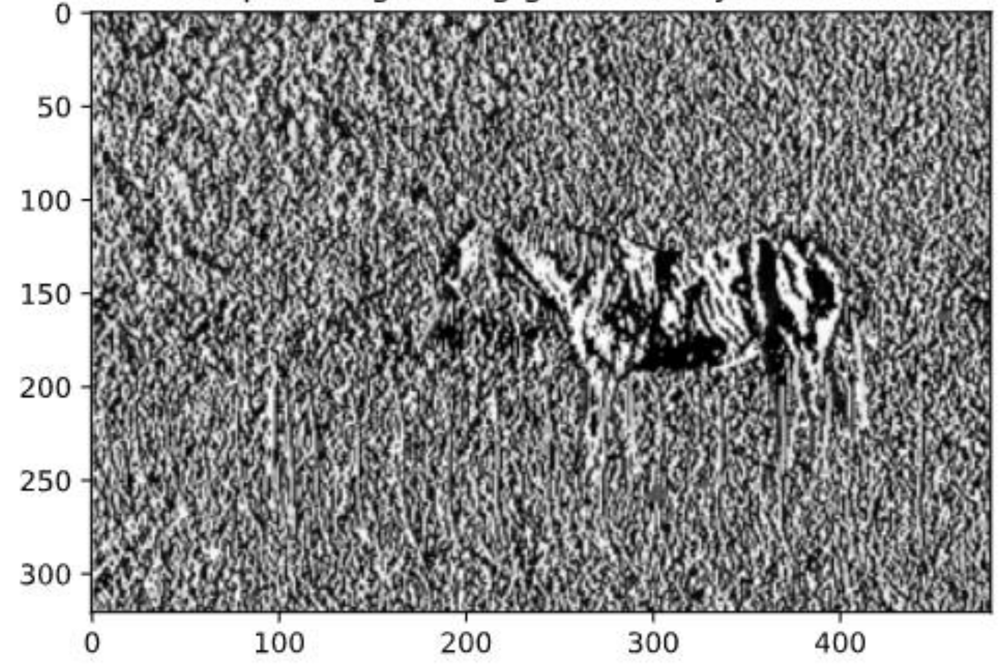
$$\frac{d}{dx}I = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } \frac{d}{dy}I = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gradient magnitude: $G = \sqrt{(\frac{d}{dx}I)^2 + (\frac{d}{dx}I)^2}$

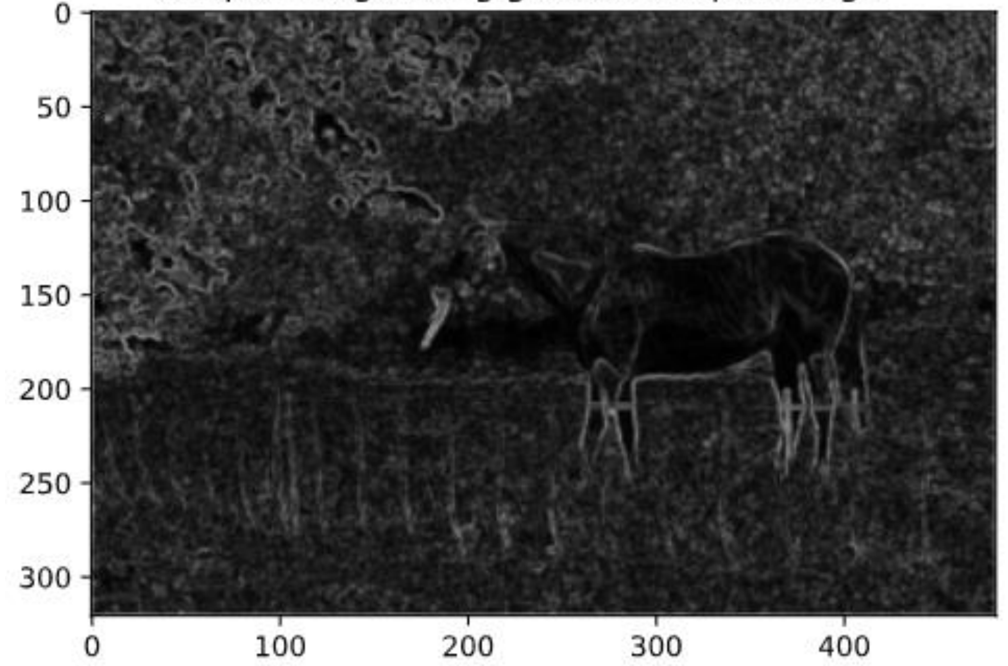
Results:

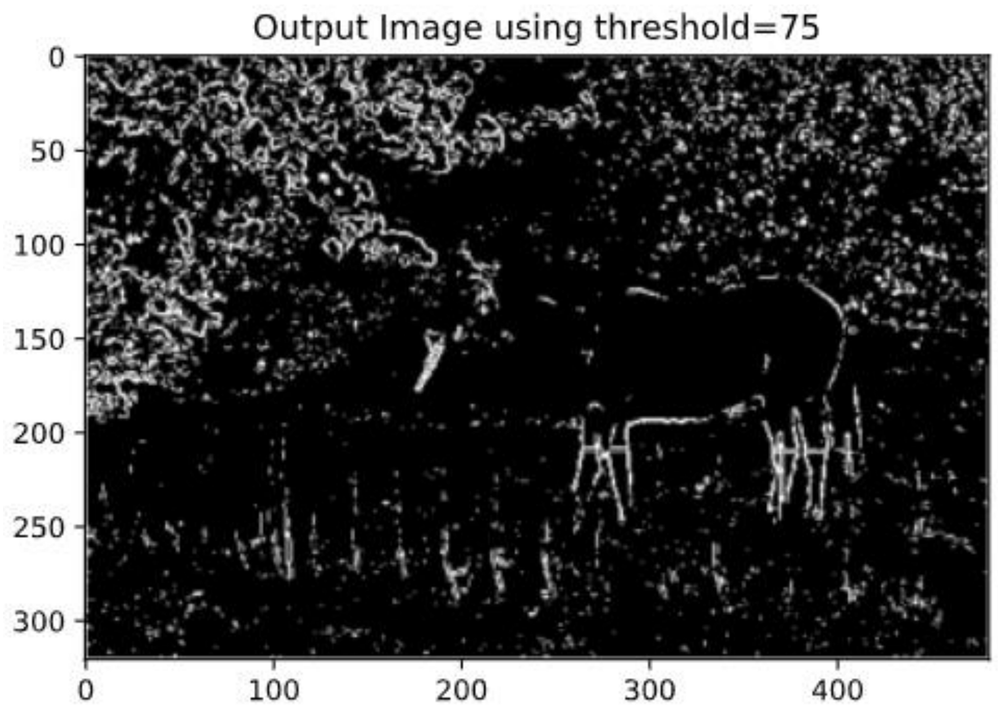


Output Image using gradient in y-direction



Output Image using gradient output image:



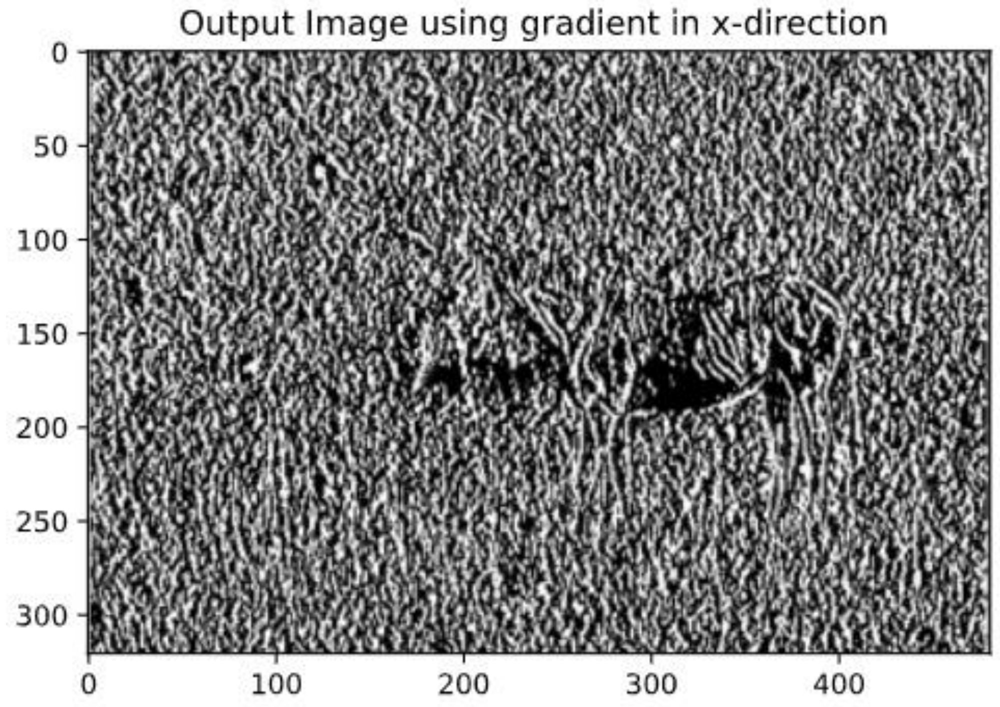

Output Image using threshold=75
0
50
100
150
200
250
300
0
100
200
300
400

Prewitt Edge Detector

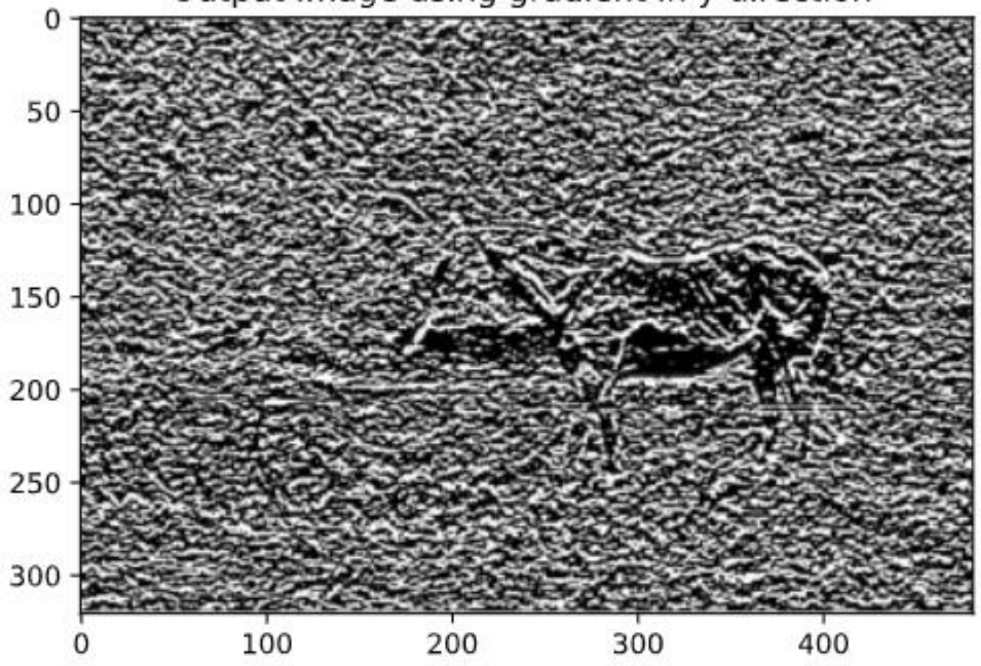
The idea is the same but different in the gradient matrices. In Prewitt Detector, we don't emphasize the middle value like in Sobel Detector.

$$\frac{d}{dx}I = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \frac{d}{dy}I = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

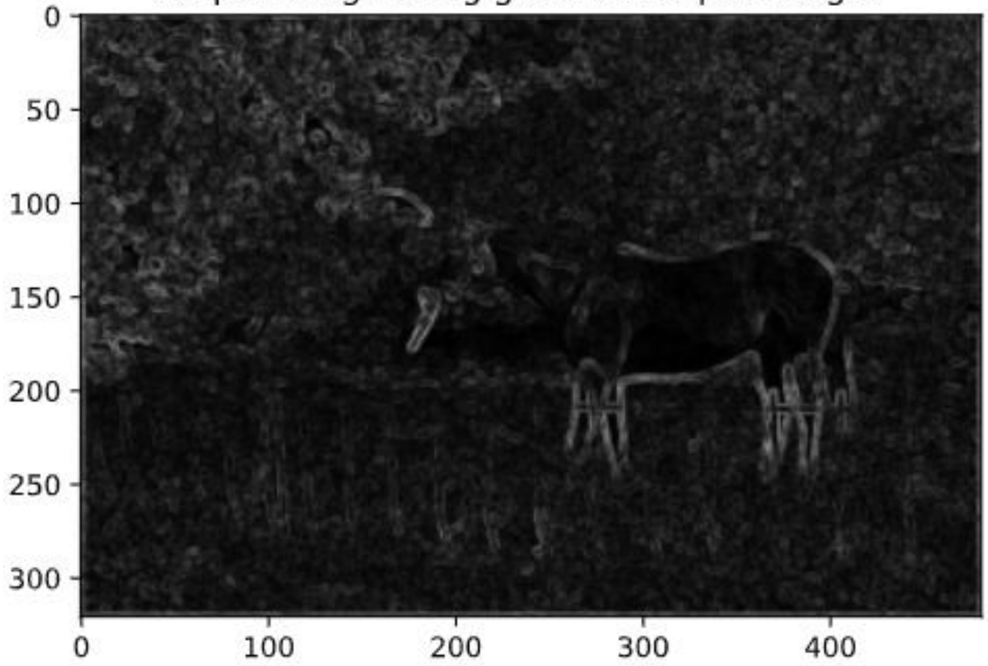
Results:

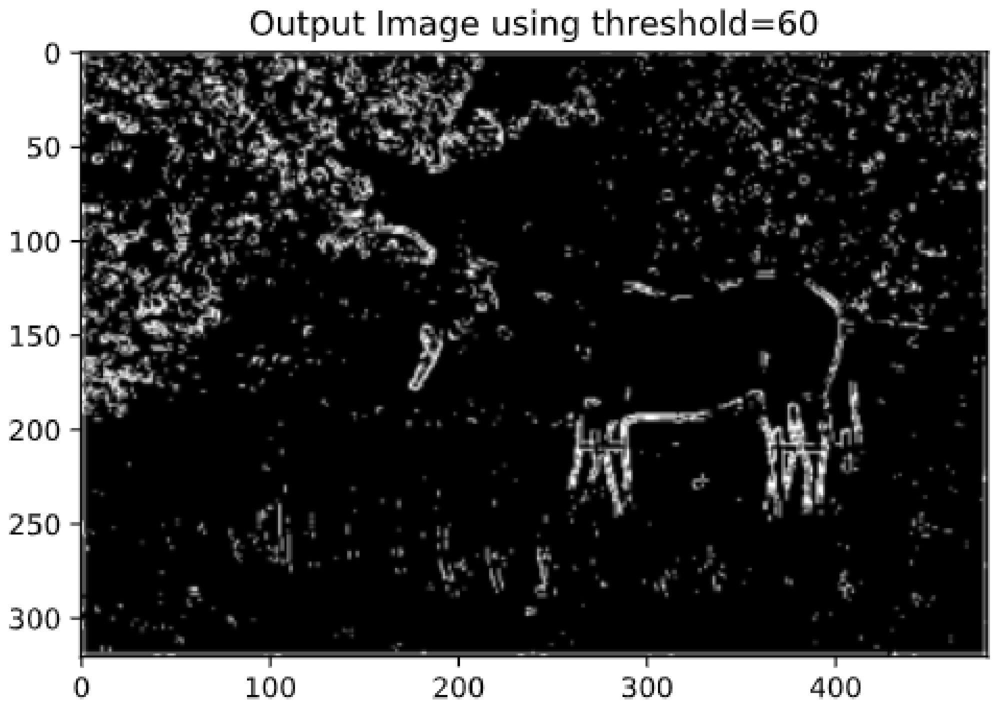


Output Image using gradient in y-direction



Output Image using gradient output image:





Comments:

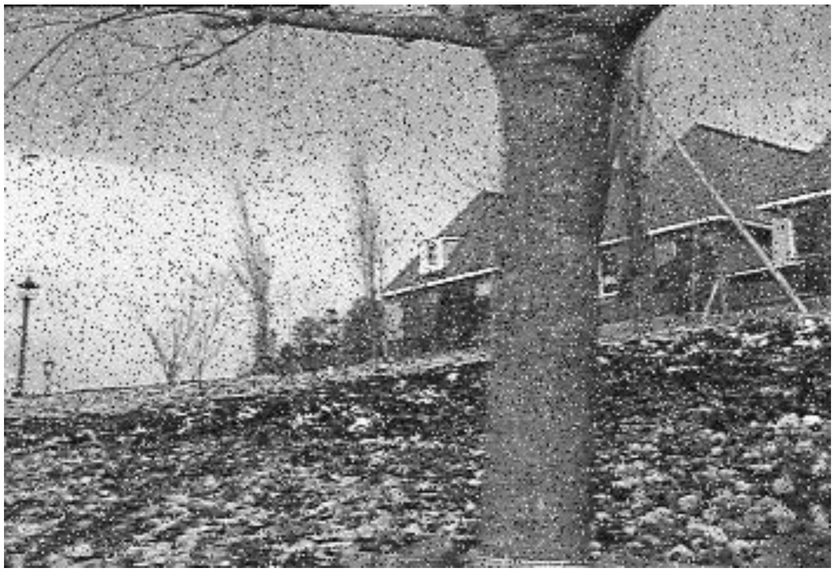
As we can see, because the Prewitt detector did not emphasize the middle value, in the x and y gradient map the filter did not recognize the horse pattern as the Sobel did. It can be explained that in the horse image there are lots of edge changes and Prewitt Detector has failed to emphasize those.

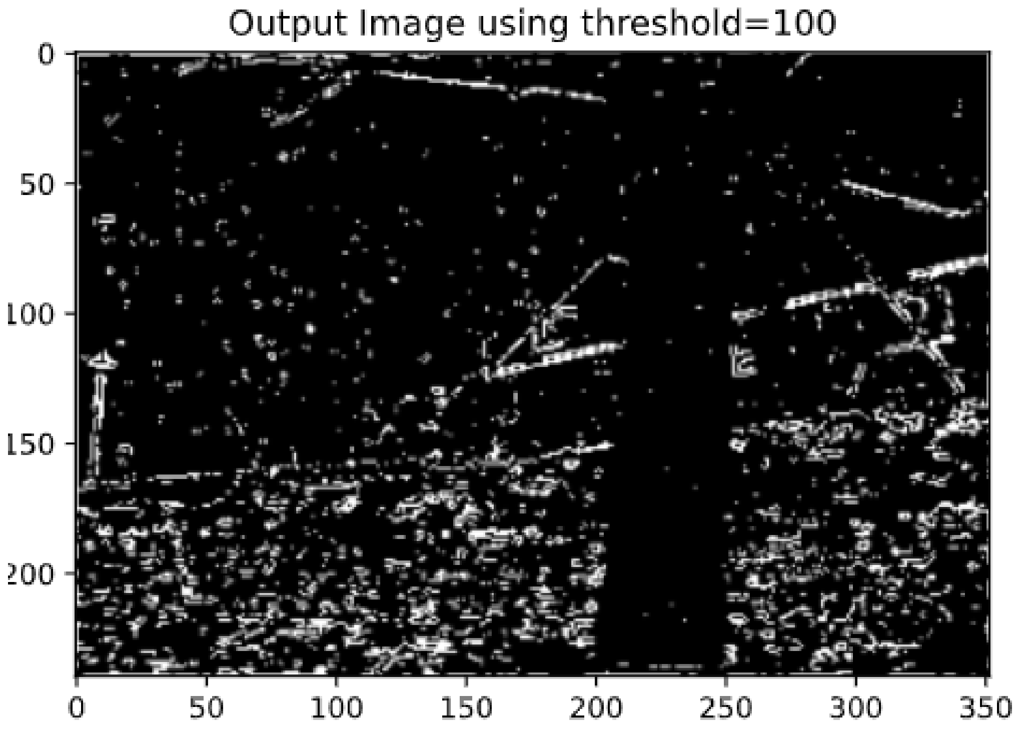
It also can be observed that in both results there is lots of noise and it clearly affects the final result.

Canny Edge Detector

Because of the noises, the edge map can be affected. It is crucial to eliminate those noises before finding the edge. We can see in the below example.

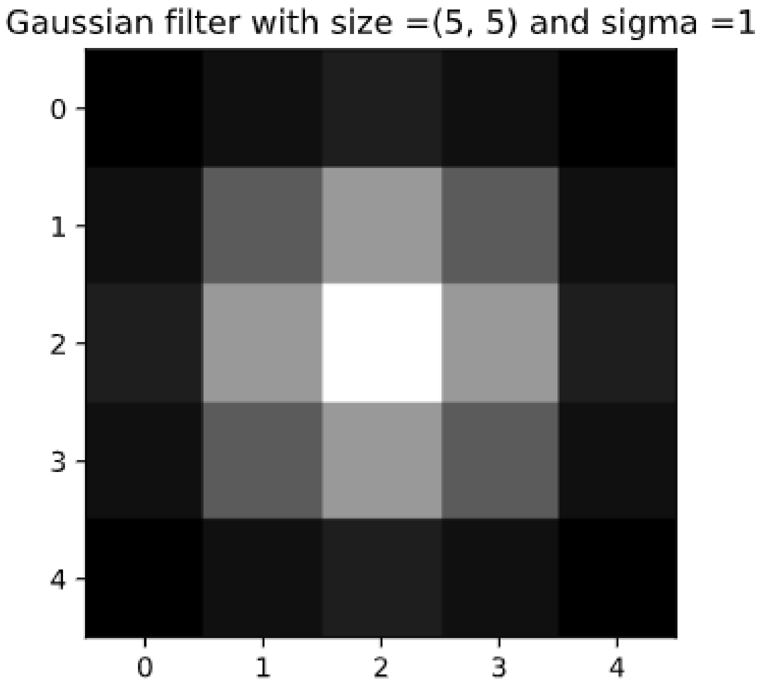
Image with noise

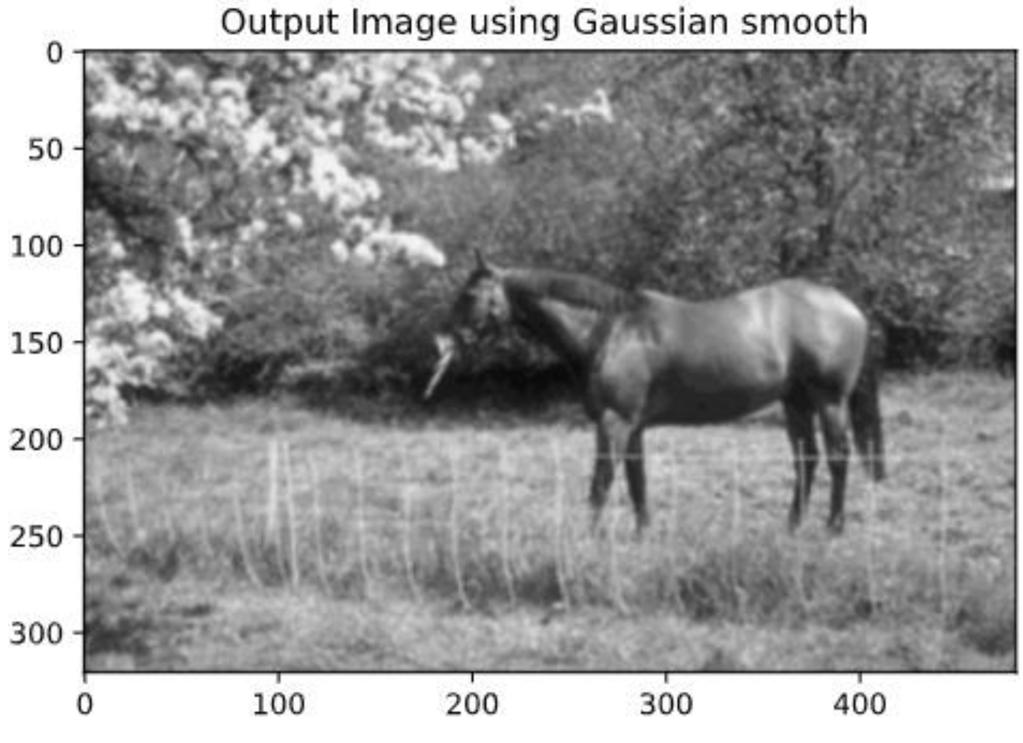




We can see that after we use the Sobel edge detector, there is lots of noise that can't be rid of. To mitigate this effect, we could use the Gaussian Filter to pre-process the image.

A Gaussian filter is just a kernel where its number is uniformly distributed.





Sigma here is to control how “fat” the kernel is going to be. Higher sigma values blur over a wider radius. Since we’re working with images, bigger sigma also forces us to use a larger kernel to capture enough of the function's energy.

Kernel = (3,3)

Sigma = 1

Sigma = 2

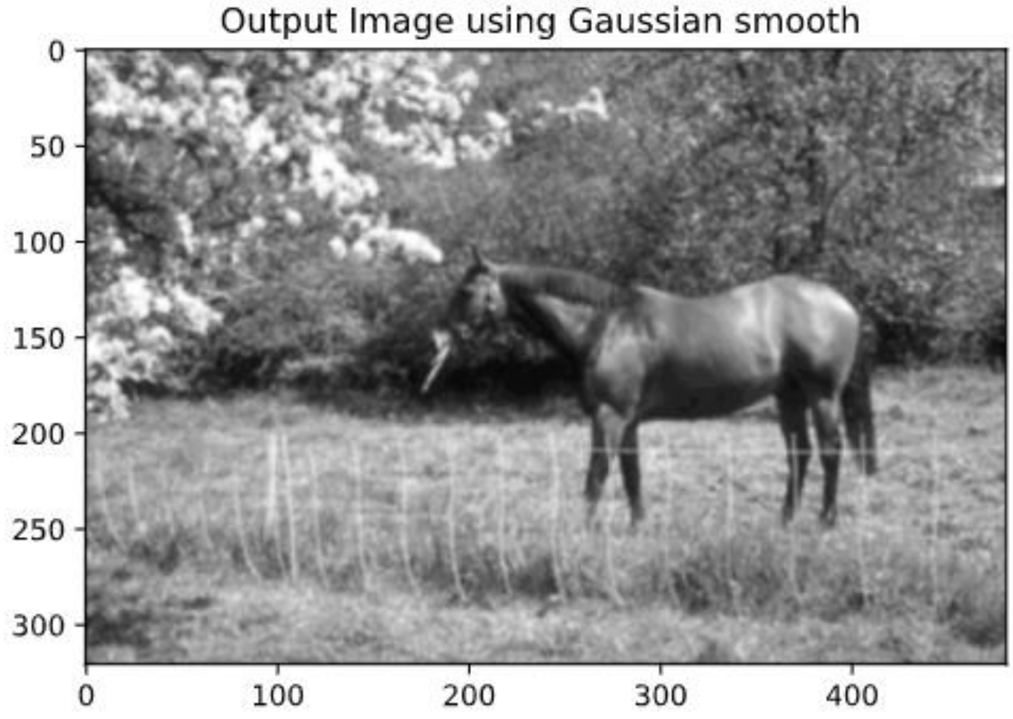
After we use the Gaussian filter to smooth out the noise, we can apply the gradient based edge detector to find the edges. As we can see in the above detectors, the edge map is not uniform. To deal with it, we could use a non-maximum suppression algorithm to delete the pixels which are not the local maximum. The details can be found in the slides.

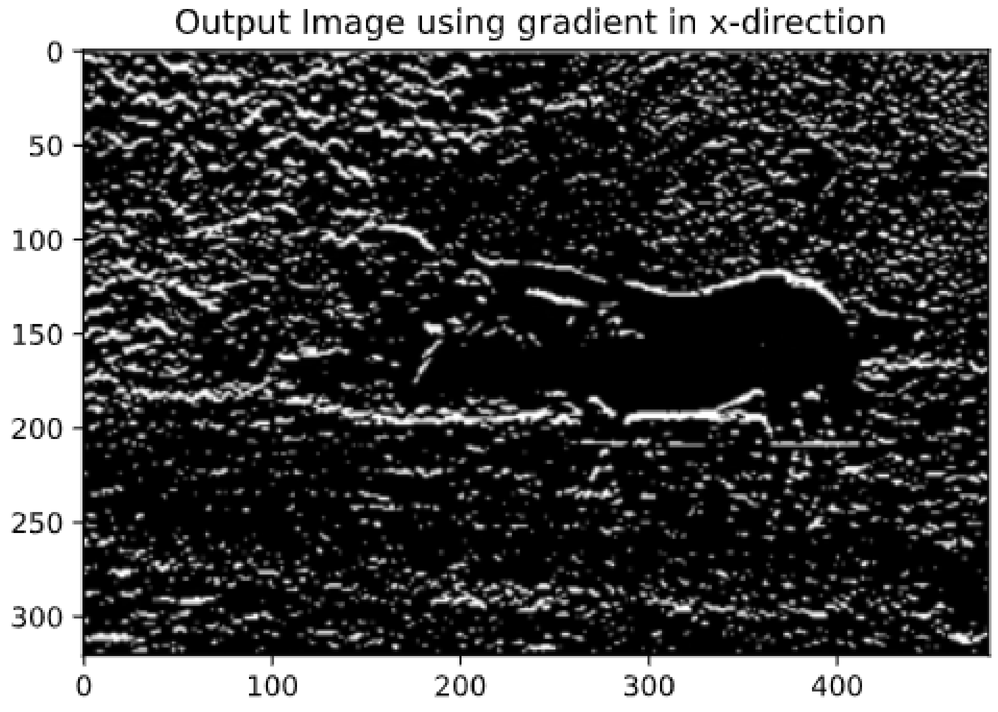
At this point, we are almost complete. We could directly use the normal threshold method to subtract all the pixels which are not edges. However, we could lose some pixels which are potentially edges. To tackle this problem, Hysteresis Thresholding is used. The main idea is that we declare High and Low pixels. If a pixel value greater than this High value, we consider it as an edge pixel, below Low, we consider it as non-edge pixel. Everything left, we need to consider its neighbors, check whether it is connected to an edge pixel. If so, we declare it as an edge pixel as well.

To sum up, Canny Edge Detector has the following steps:

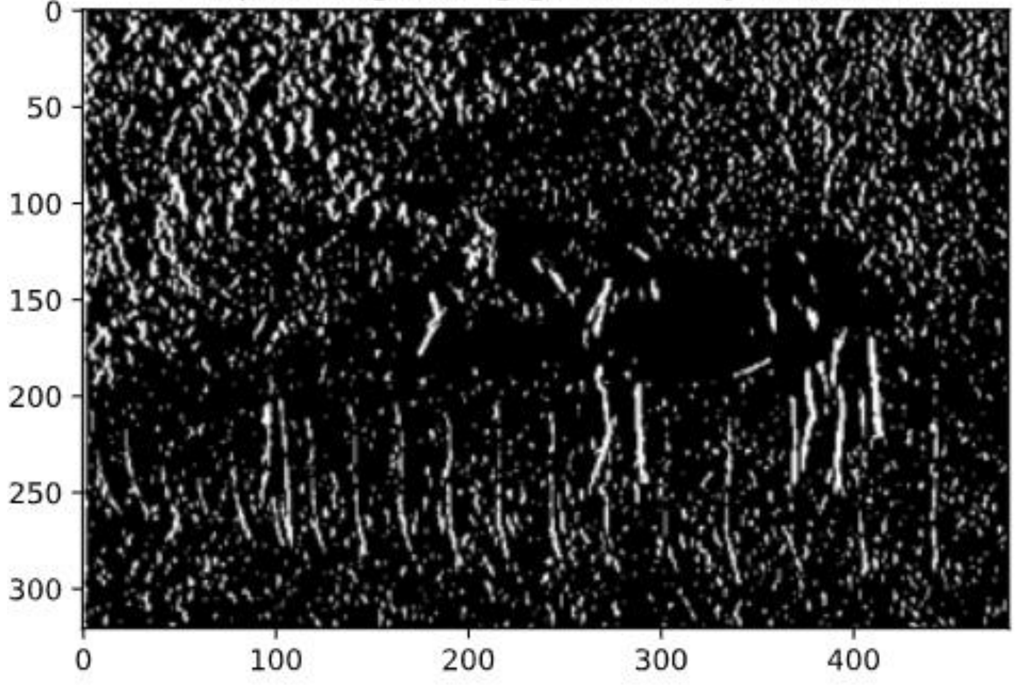
1. Gaussian filter to smooth out the noise
2. Apply gradient-based edge detector to compute the gradient magnitude and direction
3. Apply Non-maximum Suppression algorithm to suppress the pixels which are not local maximum
4. Apply Hysteresis Thresholding

Result:

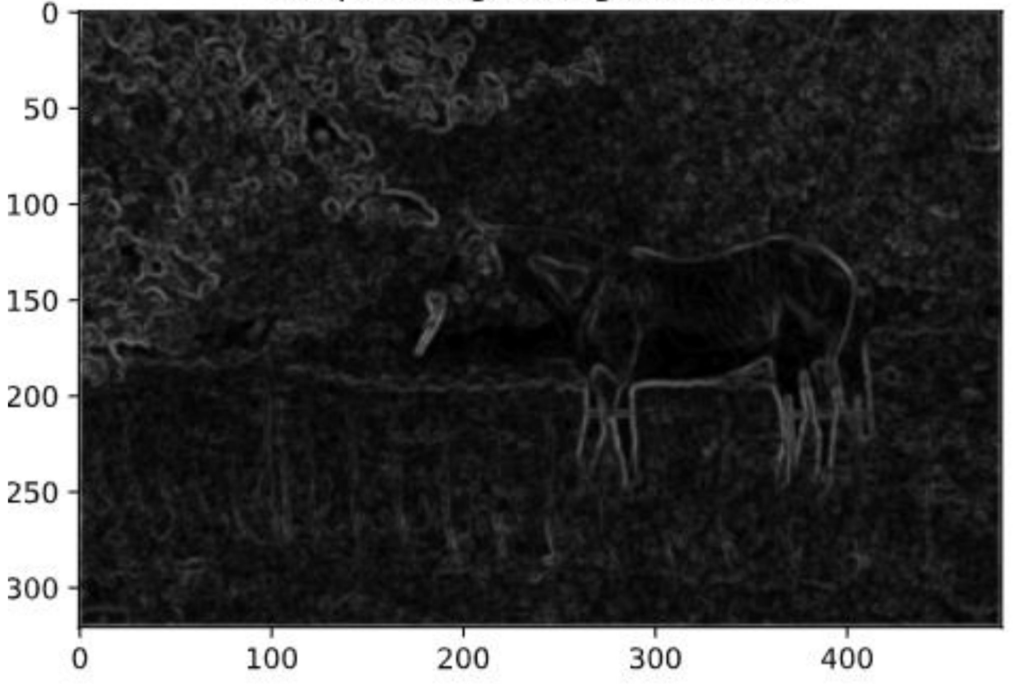

Output Image using Gaussian smooth
0
50
100
150
200
250
300
0
100
200
300
400


Output Image using gradient in x-direction
0
50
100
150
200
250
300
0
100
200
300
400

Output Image using gradient in y-direction

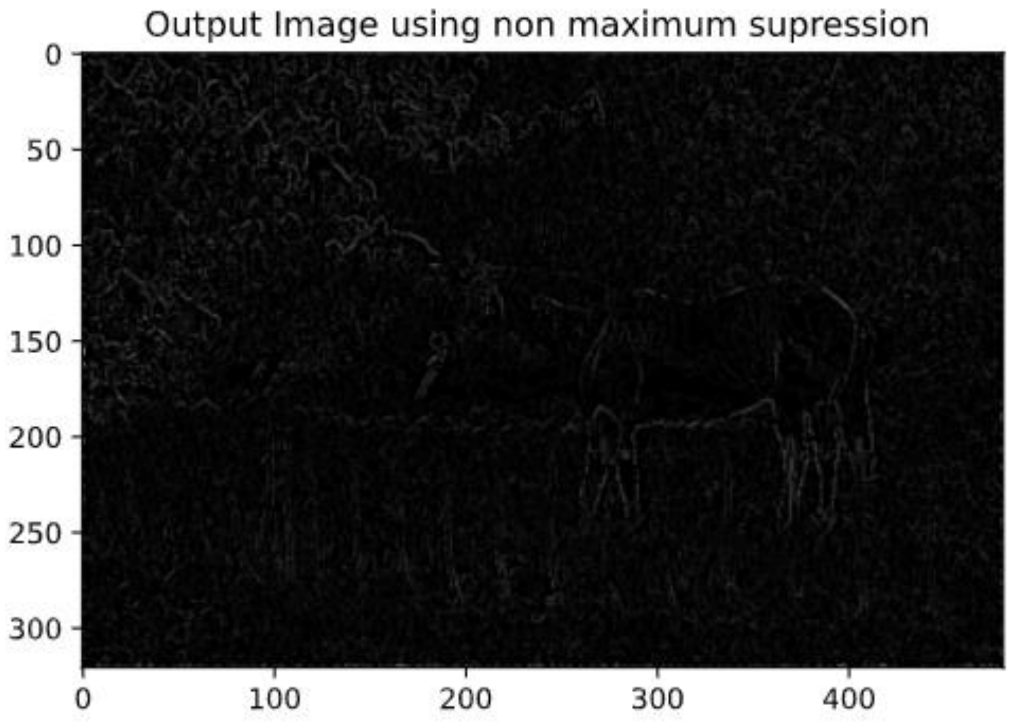


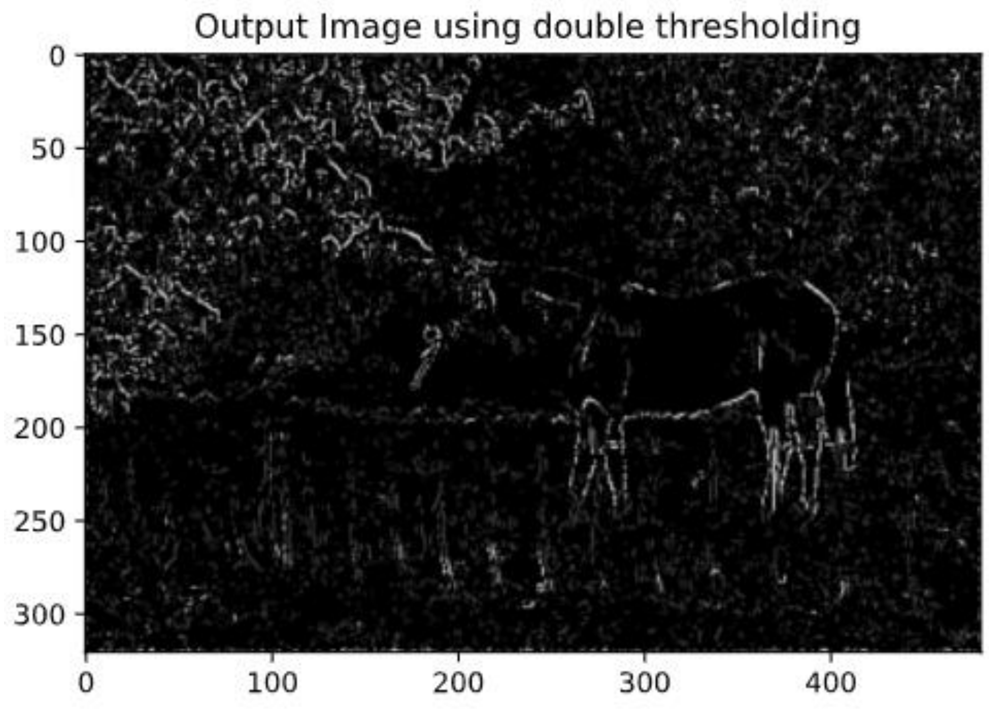
Output Image using sobel filter

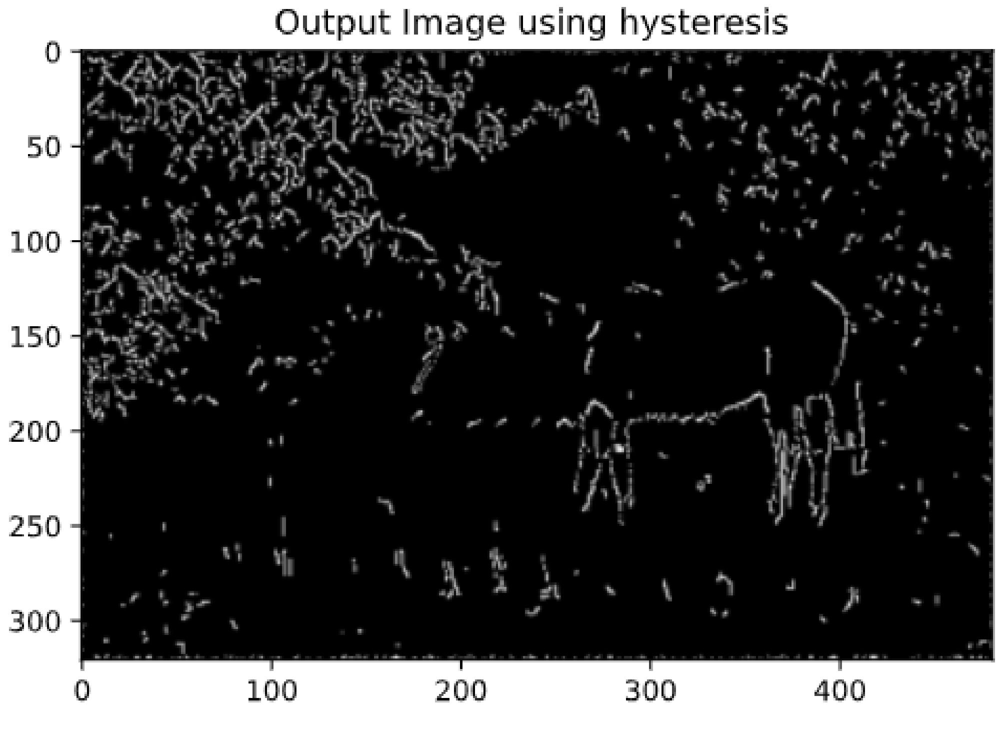


Up to this part, I discover that my Non-maximum suppression algorithm may not work correctly as it suppresses all the major edges, so I decide to show the result in two parts, with and without NMS algorithm.

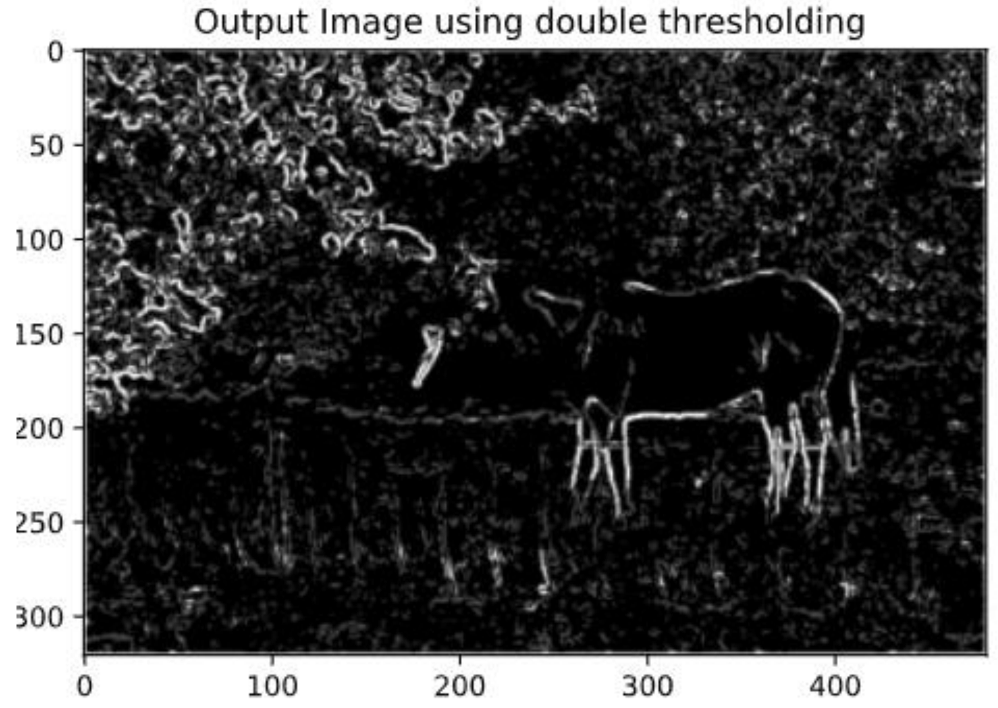
- ***With NMS:***



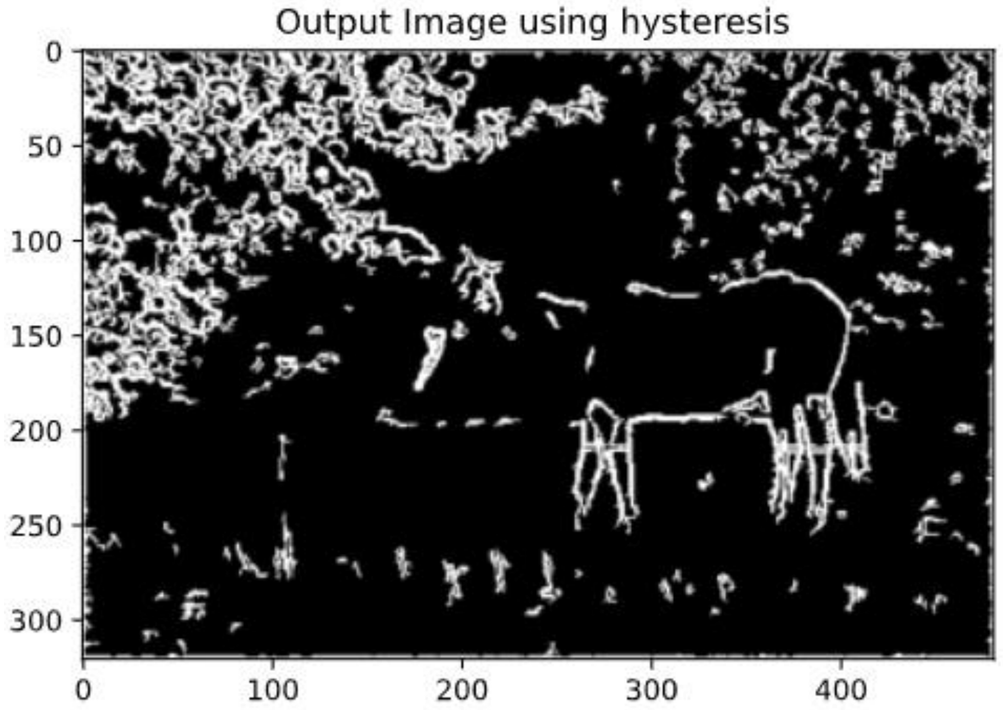

Output Image using double thresholding
0
50
100
150
200
250
300
0
100
200
300
400


Output Image using hysteresis
0
50
100
150
200
250
300
0
100
200
300
400

- ***Without NMS:***



Output Image using double thresholding
0
50
100
150
200
250
300
0
100
200
300
400



Output Image using hysteresis
0
50
100
150
200
250
300
0
100
200
300
400

Comments:

As we can clearly see, my NMS algorithm suppresses almost the major edges, it may be the image is so small, so it thins out process took out almost the edges. It is expected the canny detector has done the better job than gradient-based detectors because it eliminates almost the noise and with the usage of hysteresis thresholding, it can delete pixels which are not edges. Overall, better performance can be observed in Canny Edge Detector and it can achieve the three goals.

Final thoughts

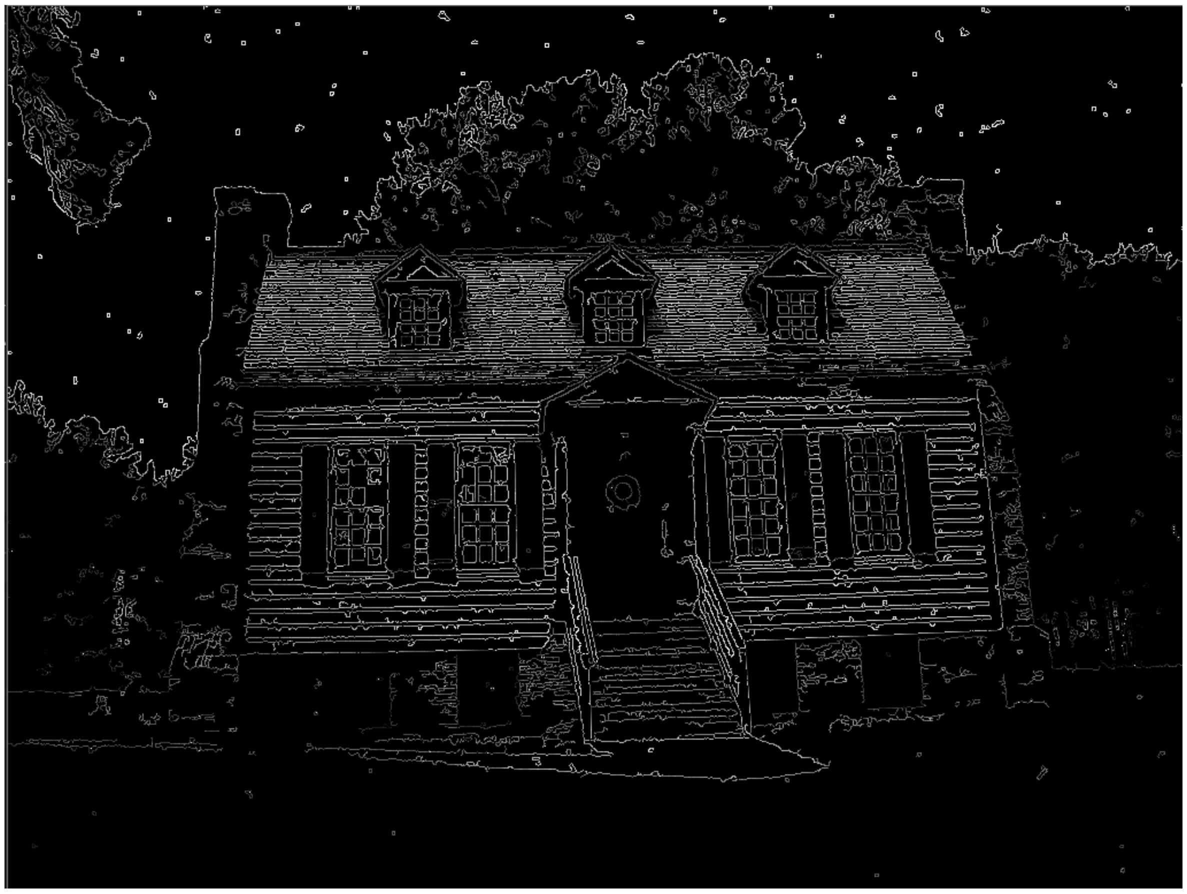
I believe that my code is not optimized because it uses the two naïve loops during most of the function, which can be improved by using matrix indexing to boost the performance. Moreover, the non-maximum algorithm tends not to work in small images.

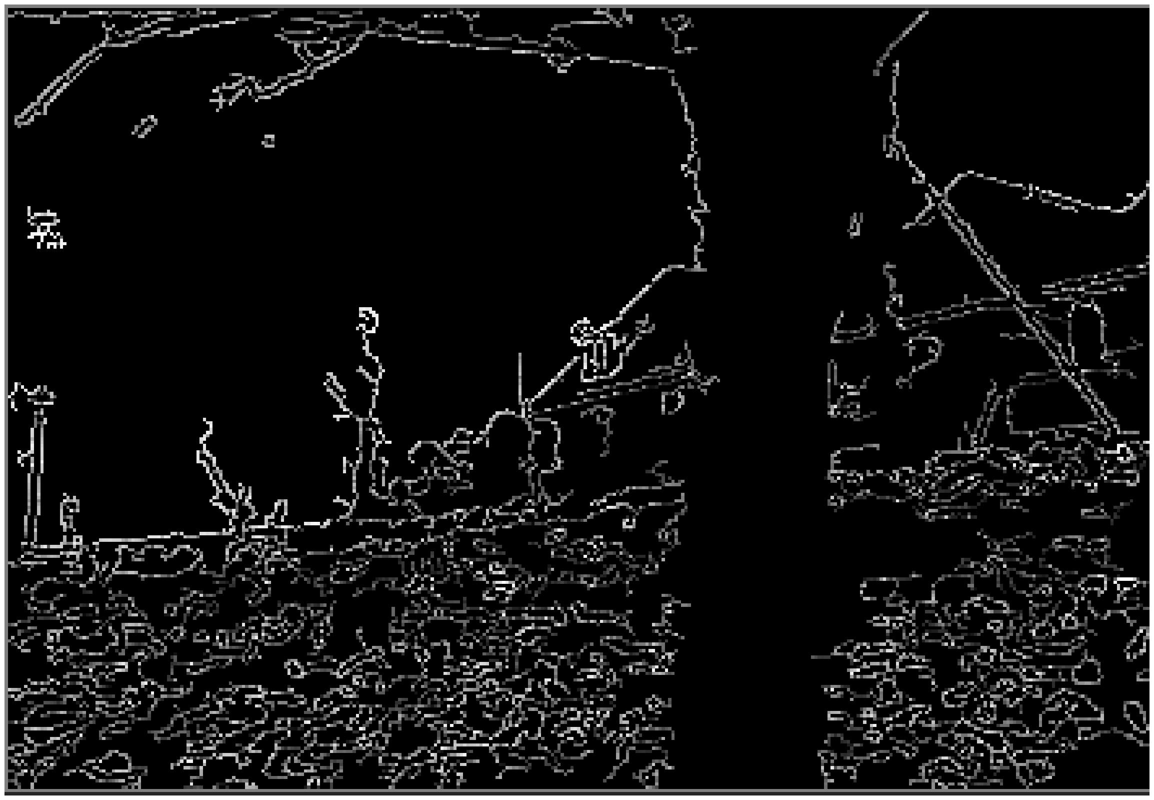
It also be seen that my implementation was still not good enough compared to other libraries. The problem might lie in the convolve function. It is needed to be investigated in the future.

Another thing is this Canny Detector depends heavily on tuning lots of parameters, for example gaussian kernel size, sigma, High pixel threshold and Low pixel threshold, etc. I believe that there is a way to automatically tune or limit the parameters needed to be tuned.

The convolve function you find in this code book is the function I already submitted in the last homework, however in the last one it ran incorrectly and generated some weird image with convolution. In this codebook, I fixed it.

I also found out some libraries already implemented Canny Detector, specifically OpenCV. I tried to run it on the images you gave me and here some results.





And here some results from my detector: