# HW5 Report

## Problem Description

In this homework, we will be implementing the Canny edge detection technique to detect edges in images. The Canny edge detection technique involves several steps. First, we need to convert the image to grayscale, to work with a single channel. Then, we apply Gaussian smoothing to reduce noise in the image. Next, we calculate the gradient of the image using the Sobel operator. This will give us the magnitude and direction of the edges. After that, we perform non-maximum suppression to thin the edges and make them more precise. Finally, we apply hysteresis thresholding to determine the final edges based on a high and low threshold value. The high threshold value determines strong edges, while the low threshold value determines weak edges. Weak edges that are connected to strong edges are considered as part of the final edge map.

## Proposed Algorithm

The implementation is done as follows:

- Gaussian_smoothing was implemented and it also utilizes the convolve function implemented in the code.
- The image gradient is calculated given the x_kernel and the y_kernel, which differ based on the method used. I won't repeat the matrices for every method because everything can be seen in the code. This part returns the magnitude and the angle of the gradient.
- Non_maximum_suppression performs non-maximum suppression on the magnitude of the gradient of an image to obtain thin edges. It takes two inputs, the magnitude and orientation of the gradient, and returns a scaled version of the non-maximum suppressed gradient magnitude. It achieves non-maximum suppression by comparing the magnitude of a pixel with its neighboring pixels along the direction of the gradient orientation and only keeping the maximum magnitude value. The output is then scaled so that the maximum value in the output is 255.
- The gradient_magnitude_histogram takes the gradient magnitude array of an image and a percentage value, computes the histogram of gradient magnitudes, normalizes it, and determines the threshold value that separates the percentage of non-edge pixels from the edge pixels, returning this threshold value as the output.
- The thresholding function takes the gradient magnitude array of an image, a low threshold value, and a high threshold value as inputs, and performs thresholding to convert the gradient magnitude image to a binary edge map where pixels above the high threshold are set to 255 (white), pixels between the high and low threshold are set to 128 (gray), and pixels below the low threshold are set to 0 (black).
- This function takes an input binary edge map after thresholding, and performs edge linking to connect weak edges to strong edges to obtain a final binary edge map. It does

this by looping over each pixel in the input image and if a pixel is already a strong edge (value of 255), it is copied to the output. If a pixel is a weak edge (value of 128), it checks its neighboring pixels to see if any of them are strong edges. If at least one of the neighboring pixels is a strong edge, the current pixel is set to 255 in the output.
This could have been done recursively, but I chose to do it iteratively as it would save space complexity and the work is very space intensive and saving on space seems like a good idea. The intuition is similar to what was done in MP1.

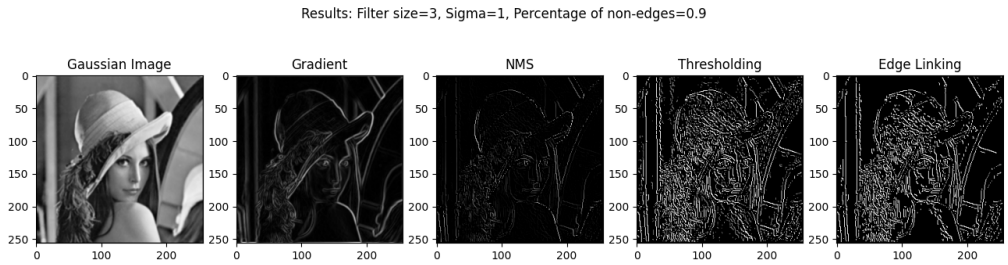
# Results

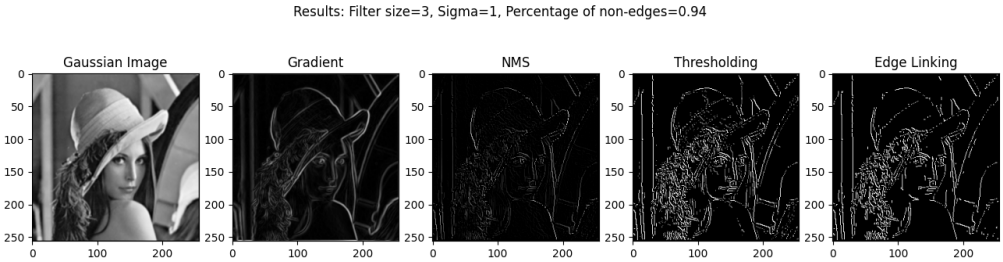
For the experiments, I varied the following hyperparameters:

- N for the gaussian smoothing taking values 3, 5, and 7
- P for percentage of edges taking values 0.9, 0.92, 0.94, 0.96
- Kernels for roberts and sobel
- Sigma for the gaussian soothing taking values 1, 2 and 3

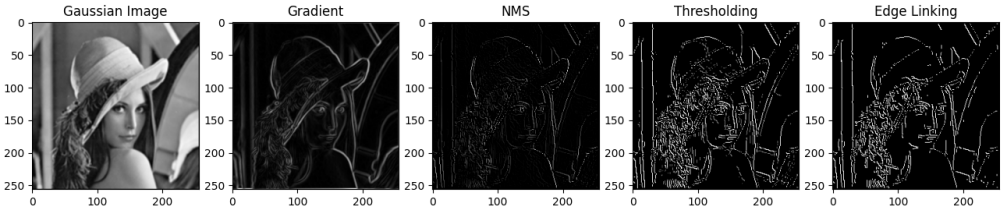
This gives a total of 3x4x2x3 = 72 different outputs for every image. It won't be feasible to show all of the results for all the images, but I will be showing 5 outputs for a few test images to see how the output changes for the different values for the hyperparameters. The values of the hyperparameters for every output can be seen at the top of all the graphs.

## Lena

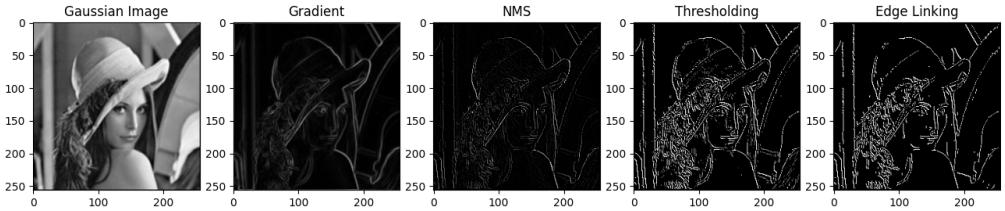


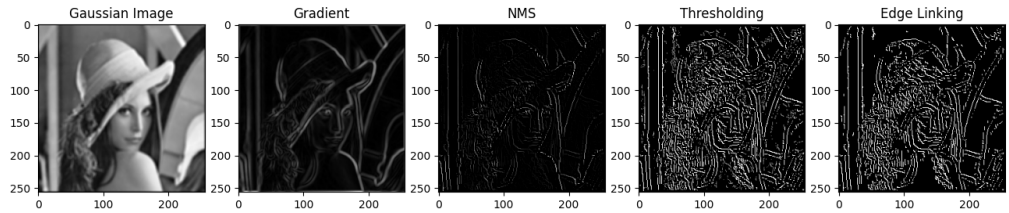
Results: Filter size=3, Sigma=2, Percentage of non-edges=0.94



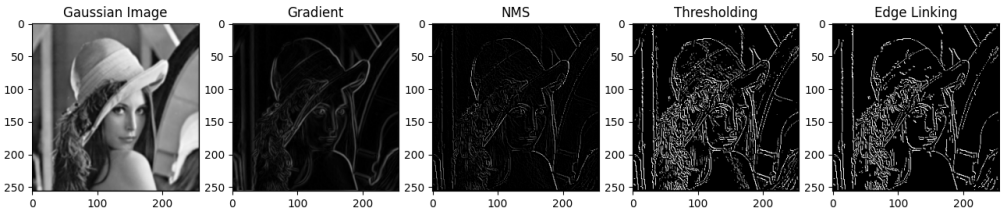

Results: Filter size=3, Sigma=2, Percentage of non-edges=0.94

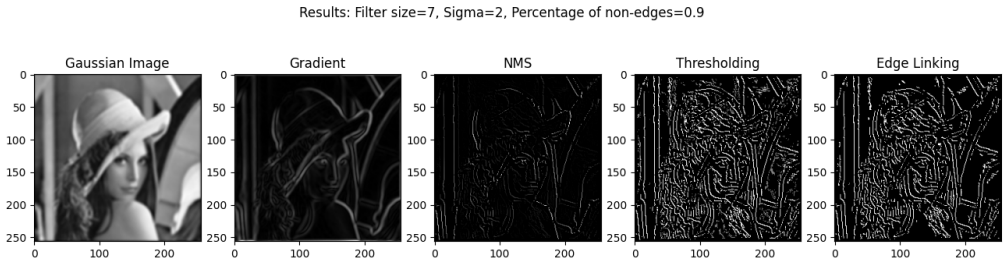



Results: Filter size=5, Sigma=3, Percentage of non-edges=0.9




Results: Filter size=5, Sigma=1, Percentage of non-edges=0.92

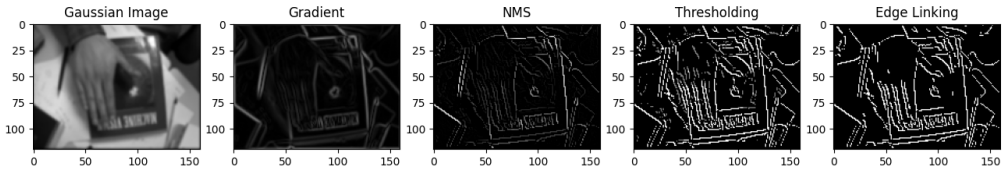




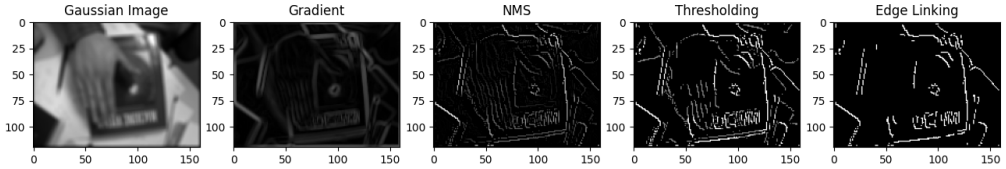

As a general impression, it feels like the canny edge detector is working as it should, and we can see a big difference in the outputs when we change the hyperparameters. Mainly, it seems like the low end values for percentage of non-edges generate a lot of additional edges, which feels unnecessary. Also, higher filter sizes for the gaussian smoothing adds a lot of noise to the image which ruins the output of the edge detector with the excess of noise. As for sigma, it seems like the ranges chosen aren't really affecting the output, and that's maybe because the values chosen are quite close which is causing minimal change in the output of the edge detector.

## Joy1

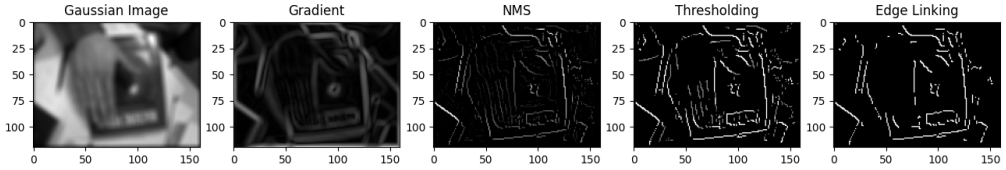
Results: Filter size=5, Sigma=1, Percentage of non-edges=0.9



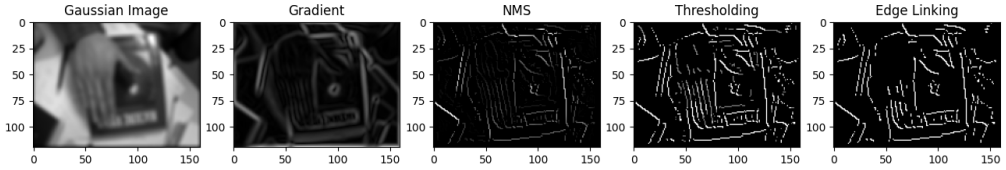

Results: Filter size=5, Sigma=3, Percentage of non-edges=0.96



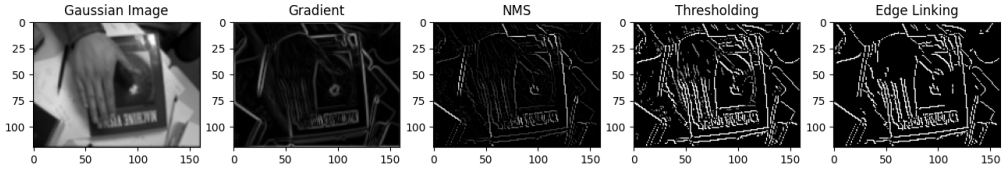

Results: Filter size=7, Sigma=3, Percentage of non-edges=0.96




Results: Filter size=7, Sigma=2, Percentage of non-edges=0.94



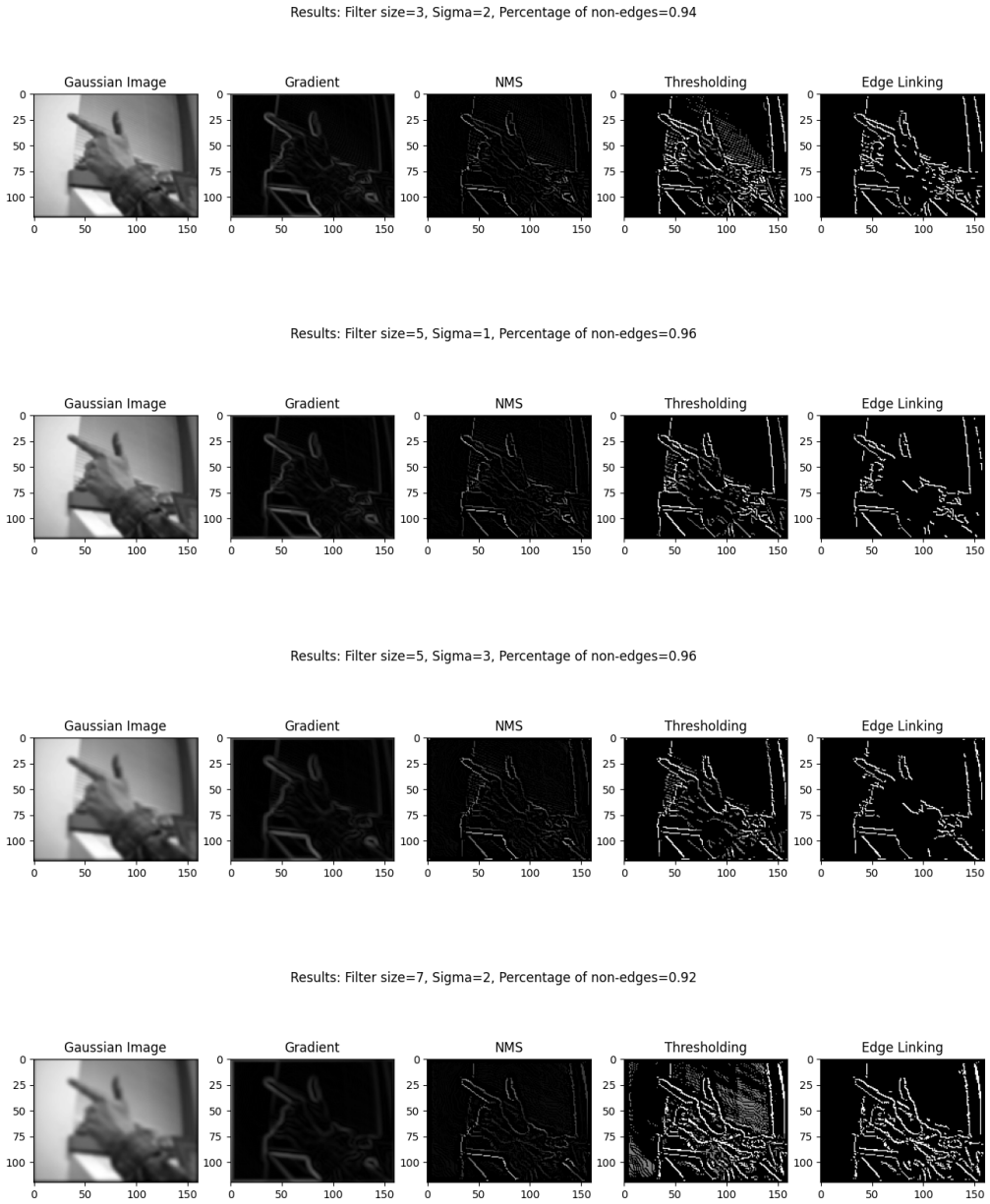

Results: Filter size=3, Sigma=1, Percentage of non-edges=0.9



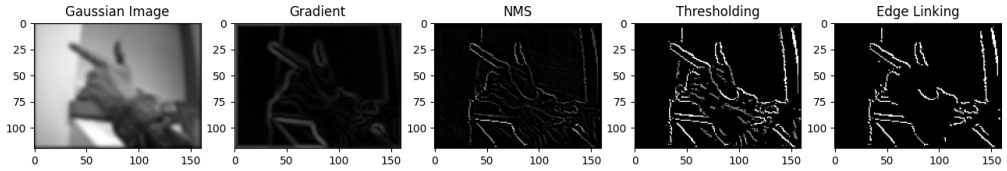

Analyzing the values we got here, it seems like lower percentage of non-edges fit this image

better than the other, and we can even say that higher values are causing the edge detector not to capture too much information and are weakening the output of the detector. The rest of the hypothesis proposed above in the lena picture seem to hold in the joy1 example.

## Gun1



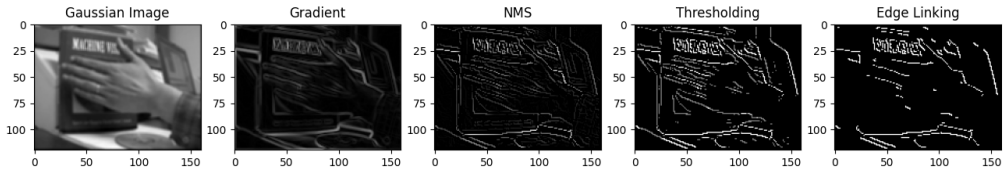
Results: Filter size=7, Sigma=2, Percentage of non-edges=0.96



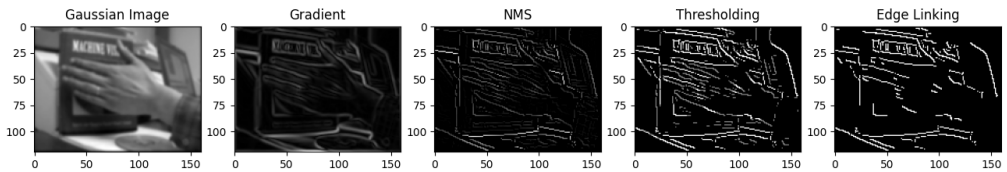

The same can be said about the gun image; the output differs for the multiple values of the hyperparameters, thus it seems like trying out different values for hyperparameters like the percentage of non-edge is a good strategy to get the desired outputs.

## Pointer1

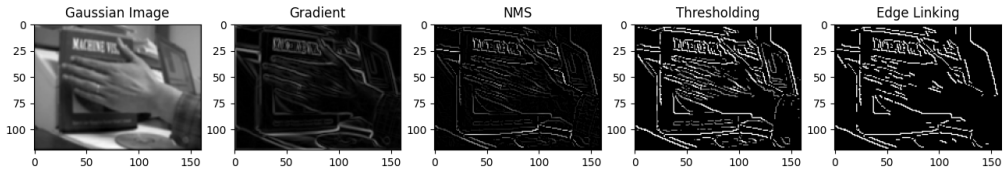
Results: Filter size=3, Sigma=2, Percentage of non-edges=0.96

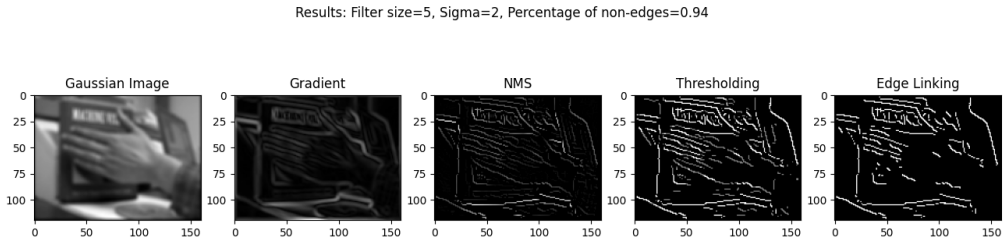


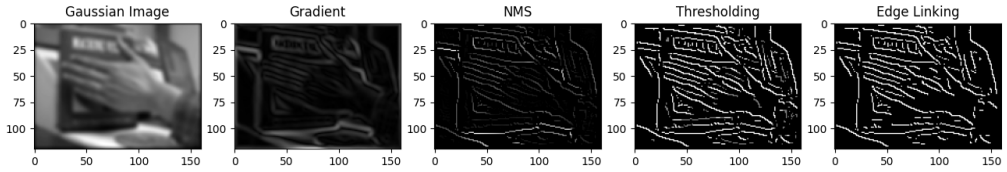

Results: Filter size=5, Sigma=1, Percentage of non-edges=0.94




Results: Filter size=3, Sigma=1, Percentage of non-edges=0.92



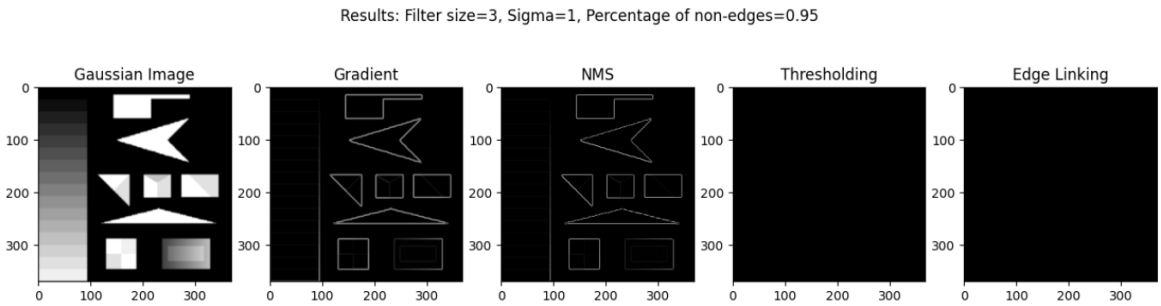


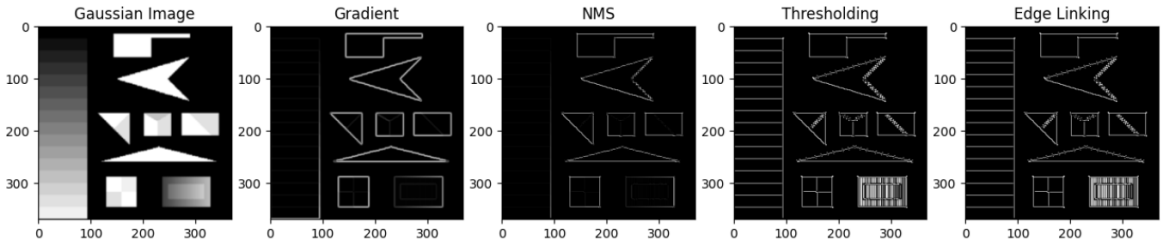

The analysis above still holds in this example.

## Test1

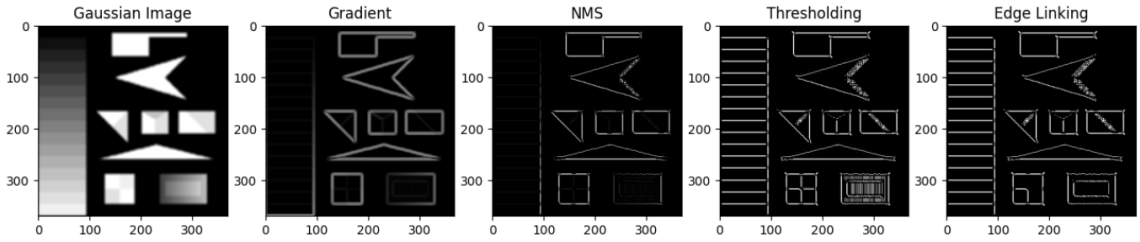
For this image, we had a less restrictive threshold of 0.99 because the edges are less clear because the gradient behaves differently in this image compared to the others, due to the fact that the image includes clear shapes with a high contrast between the background and the shape.
Here we can see some of the examples, starting with 0.95 as a lower value, going to 0.999 later. We can see that the results don't show at the beginning due to the choice of hyperparameters, but this changes as we proceed:



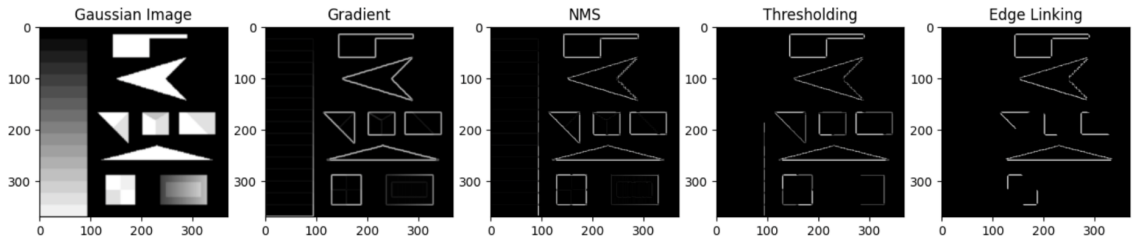
Results: Filter size=3, Sigma=3, Percentage of non-edges=0.95

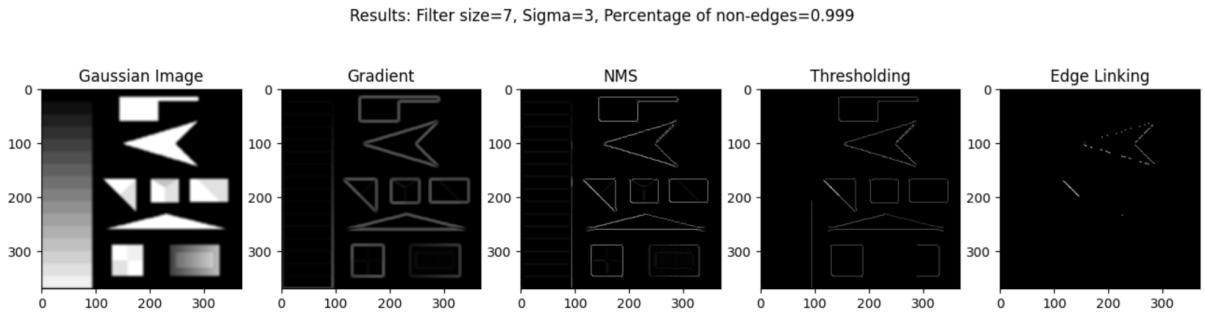


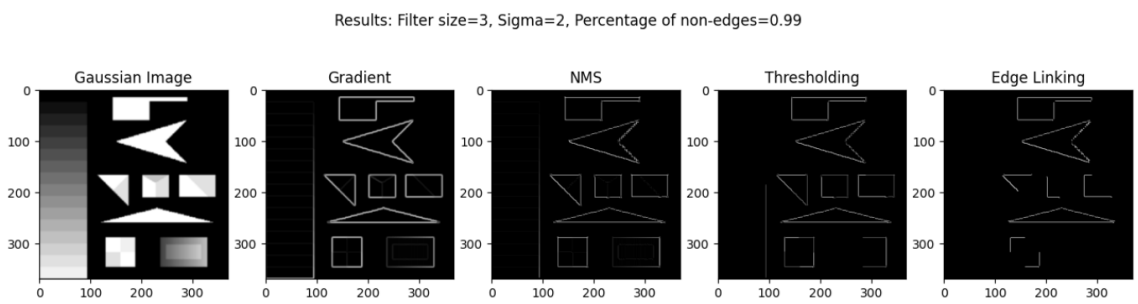

Results: Filter size=7, Sigma=3, Percentage of non-edges=0.95




Results: Filter size=3, Sigma=1, Percentage of non-edges=0.99



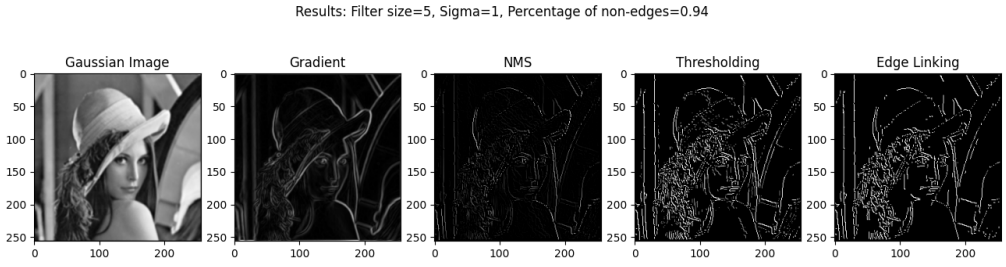


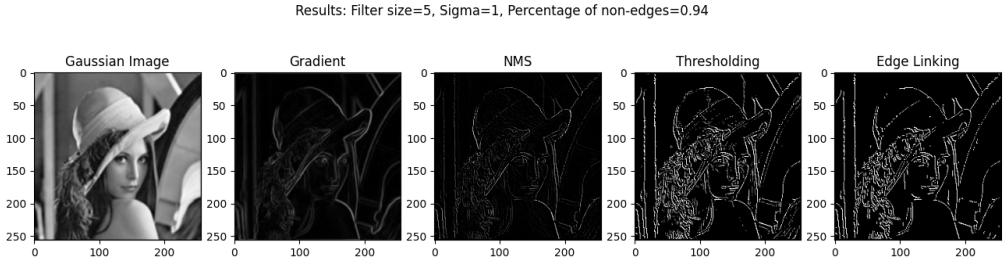

# Comparing Sobel and Roberts

## Example 1

Sobel:

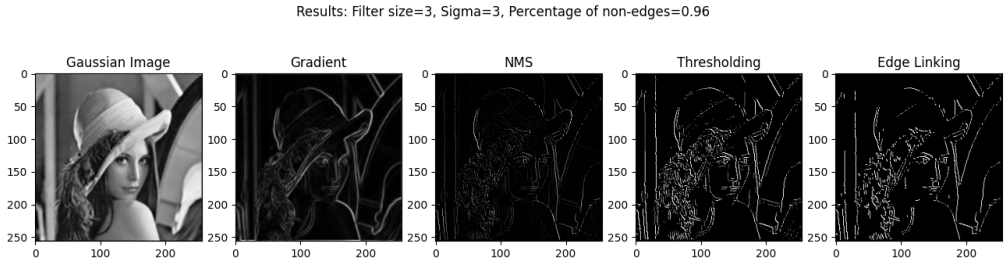



Roberts:

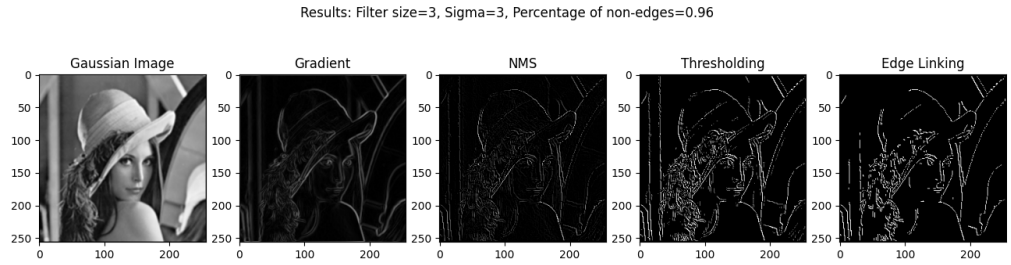


## Example 2

Sobel:

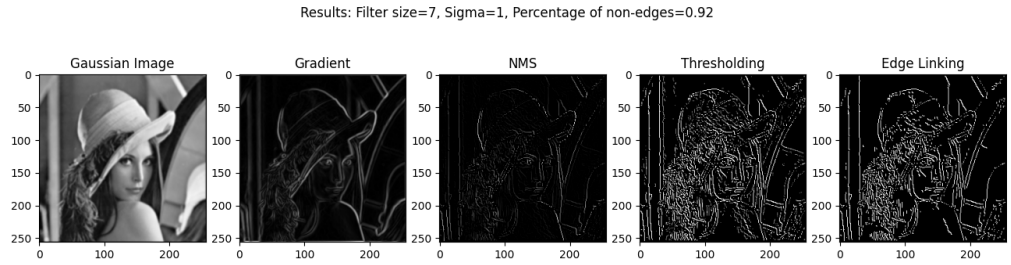



Roberts:

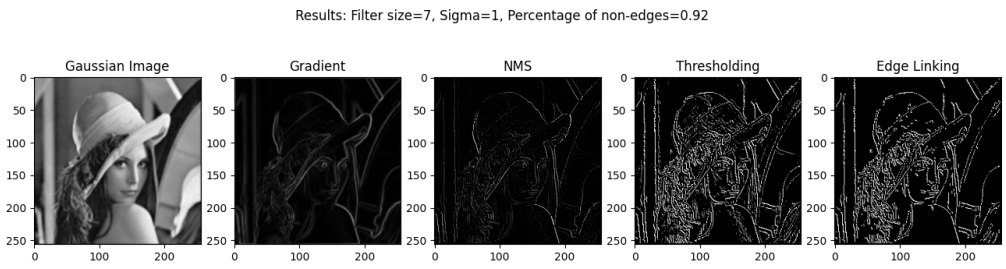



## Example 3

Sobel:




Roberts:




The results are really close and not much can be said from looking at them. We can just say that overall, the sobel filter seems to lead to more edges in the output.