CANNY EDGE DETECTOR

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

Algorithm :
Step1. Filter image with derivative of Gaussian
Step2. Find magnitude and orientation of gradient
Step3. Non-maximum suppression
Step4. Linking and thresholding(hysteresis):
- Define two thresholds : low and high
- Use the high threshold to start edge curves and the low threshold to continue them.

Observations:

Selection of Low & High Threshold values:

- As the output of the canny edge detection algorithm is highly dependent on the threshold values.There are different threshold values for each image to achieve accurate results.
- As the task is tedious to choose threshold hyperparameter values for the new image.So it is convenient to choose threshold ratio instead of specific values and generate threshold values using those ratios. Threshold ratios are set to be accordingly to get the best possible results as in my case it is 0.1 and 0.2. These same ratios are helpful in approximating the threshold values for other images to detect edges.

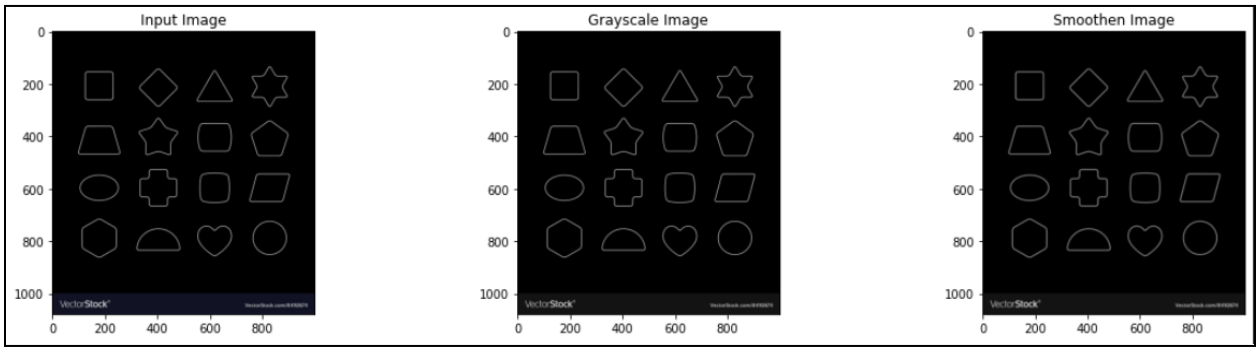
  LowThresholdRatio = 0.1
  HighThresholdRatio = 0.2

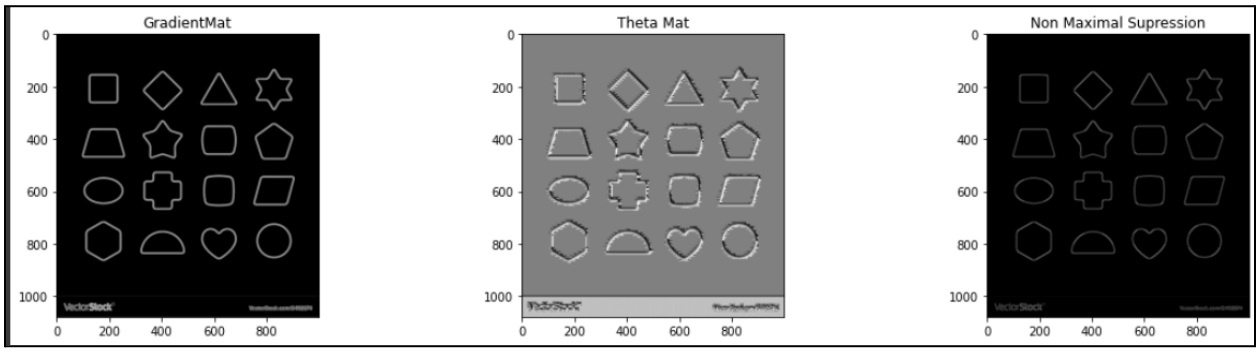
  Difference between Pixels diff = Img.max() - Img.min()
  HighThresholdRatio = diff * HighThresholdRatio
  LowThresholdRation = diff * LowThresholdRatio

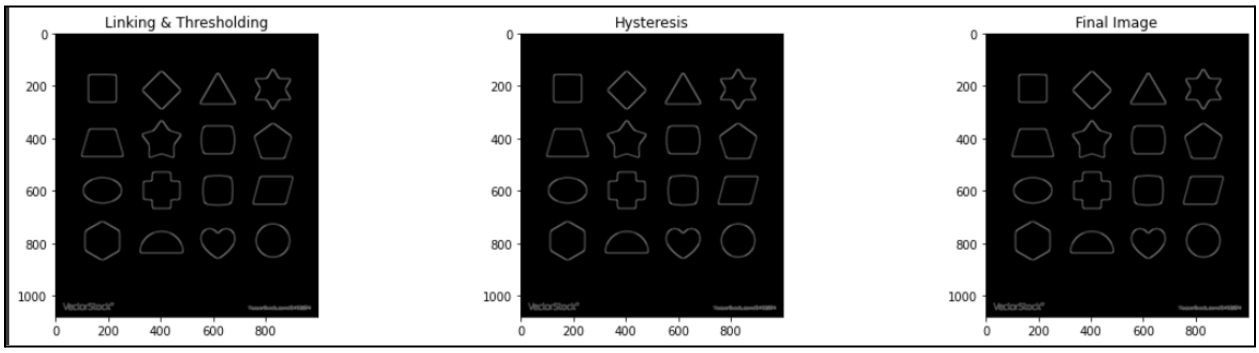
- The choice of sigma (Gaussian kernel spread/size) depends on the desired behavior as large sigma detects large scale edges and small sigma detects fine edges.
- Apply gaussian blur to smoothen the image by convolving the image with Gaussian Kernel. We can have different kernel sizes, sizes depending on the expected blurring effect. Smallest kernel means less visible blur. In our case I used a 5x5 kernel.
- In hysteresis the process is recursive as looping over the threshold image converting a lot of weak edges into true edges by checking all its neighbors.
- After converting those weak edges into true edges the final image has way more details as compared to thresholded image in step4.
- The output of the self implemented canny edge detector was nearly accurate.

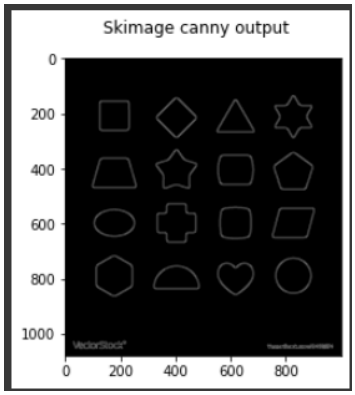
Results:

Intermediate Results of Self Implemented Canny Edge Detector

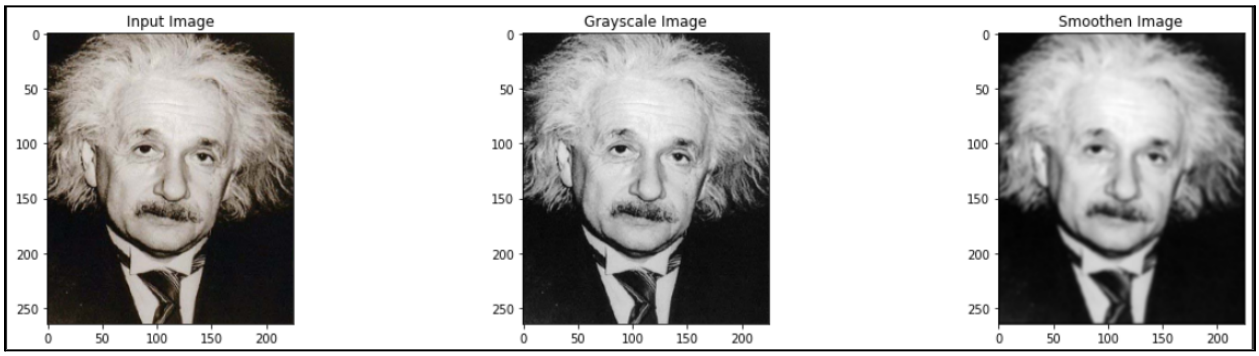


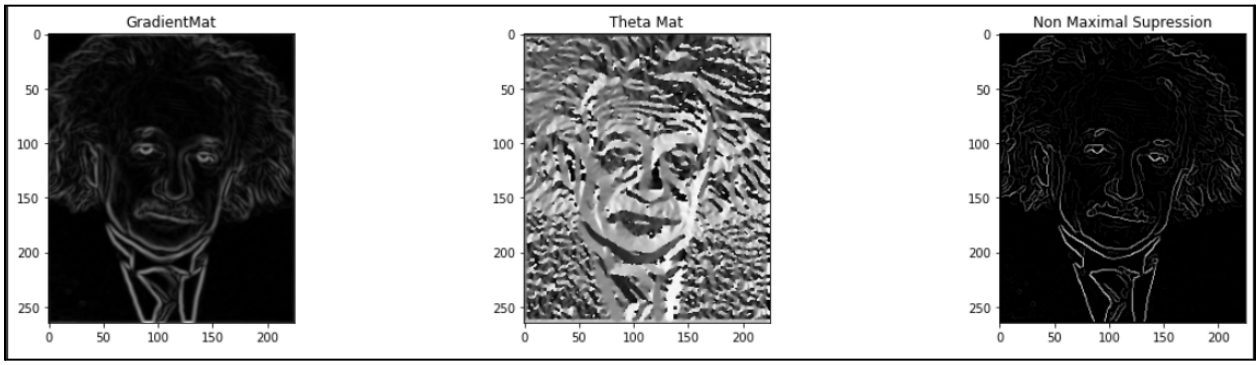


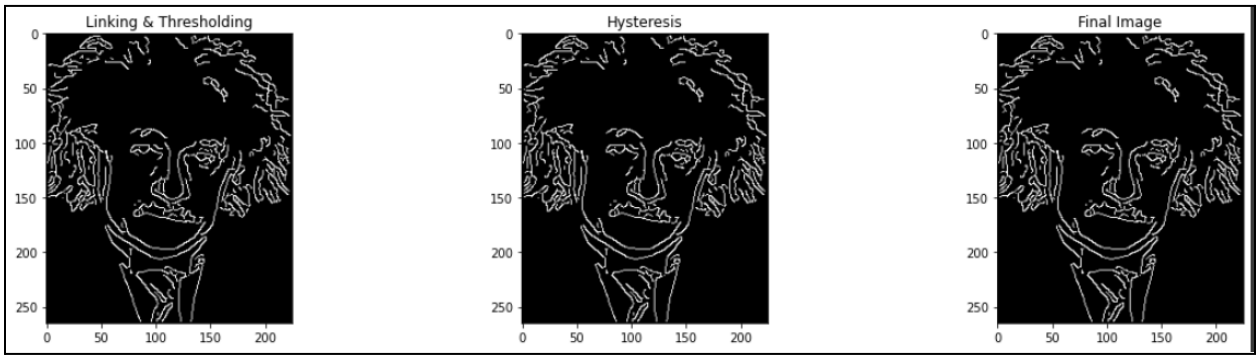


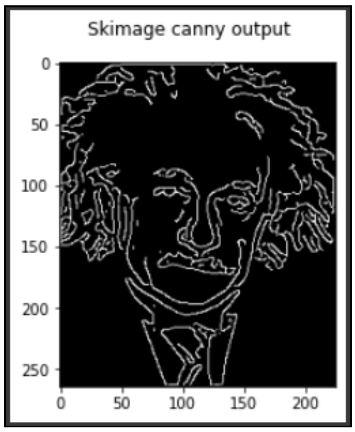


PSNR: 17.82 SSIM: 0.9201

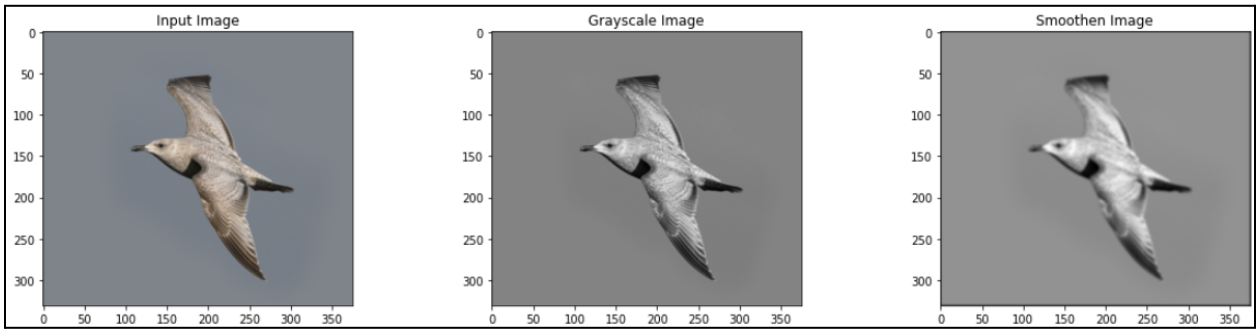


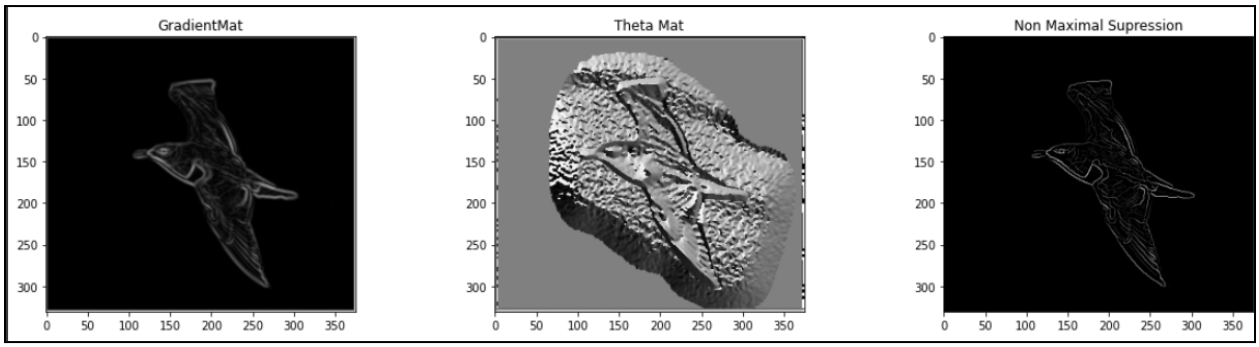


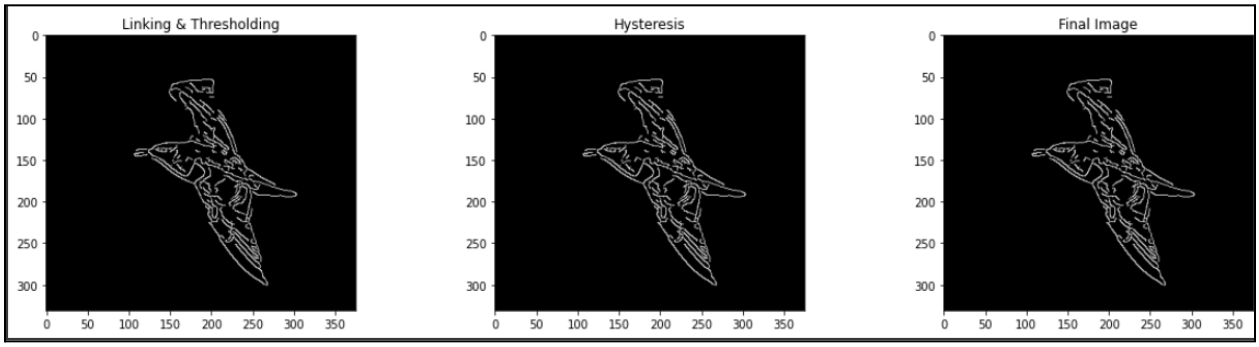


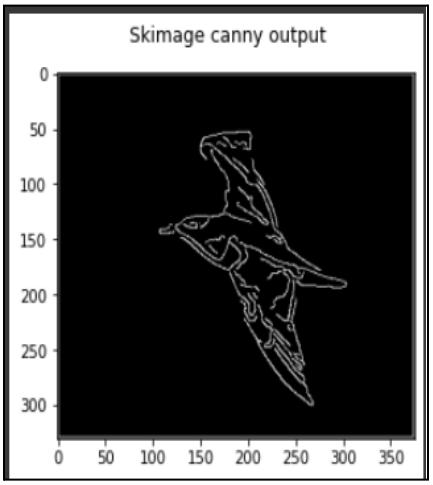


PSNR : 11.147959617274424 SSIM: 0.6075388754630748

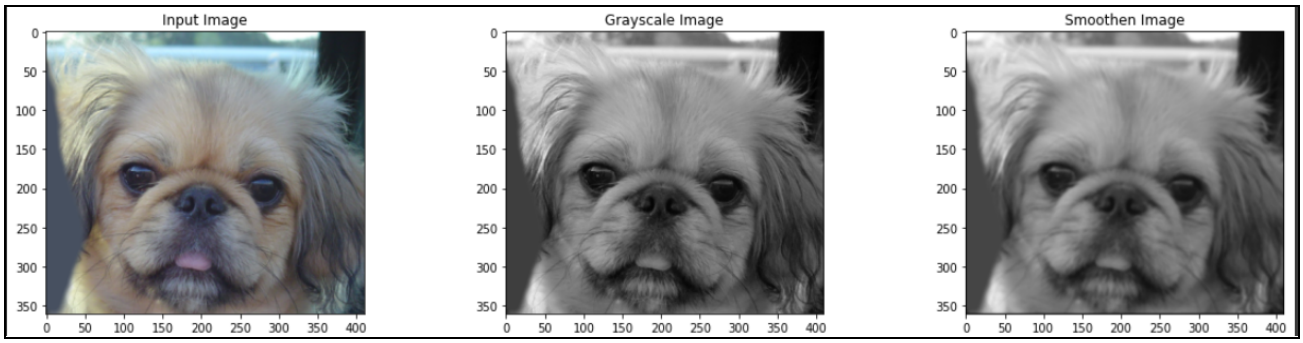


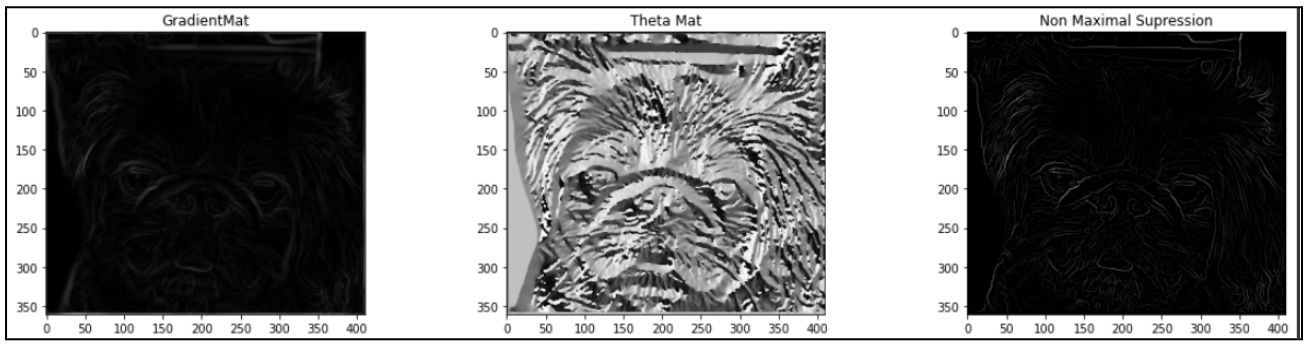


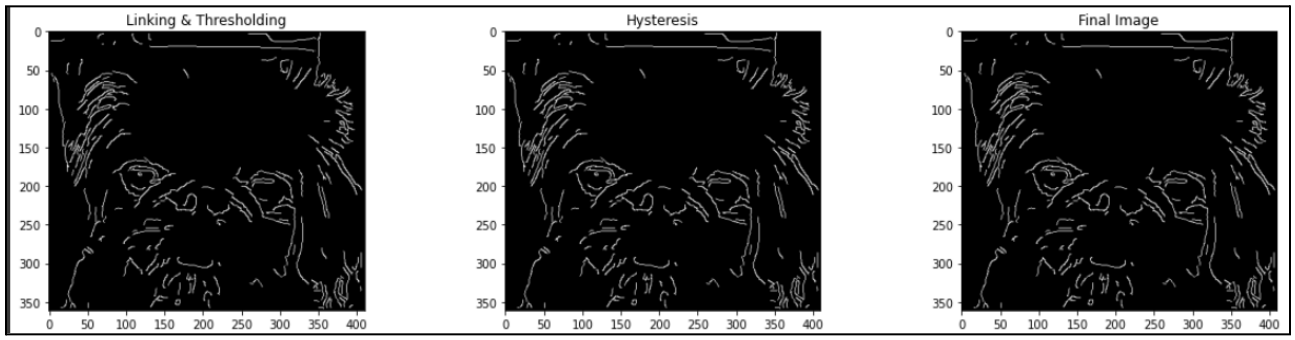


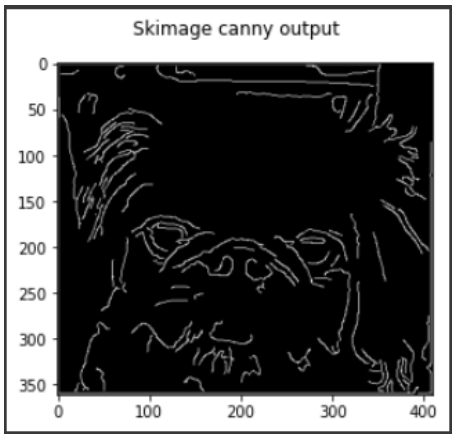


PSNR : 16.369293269452555 SSIM: 0.8649599248972781

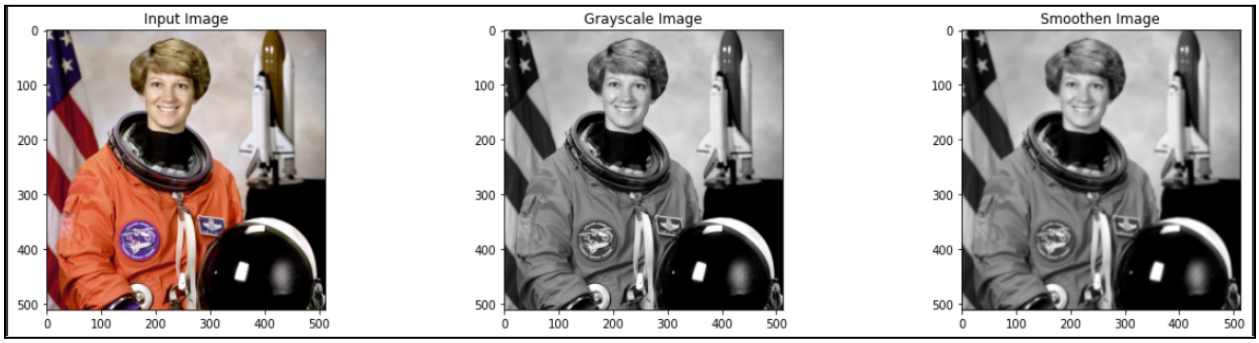


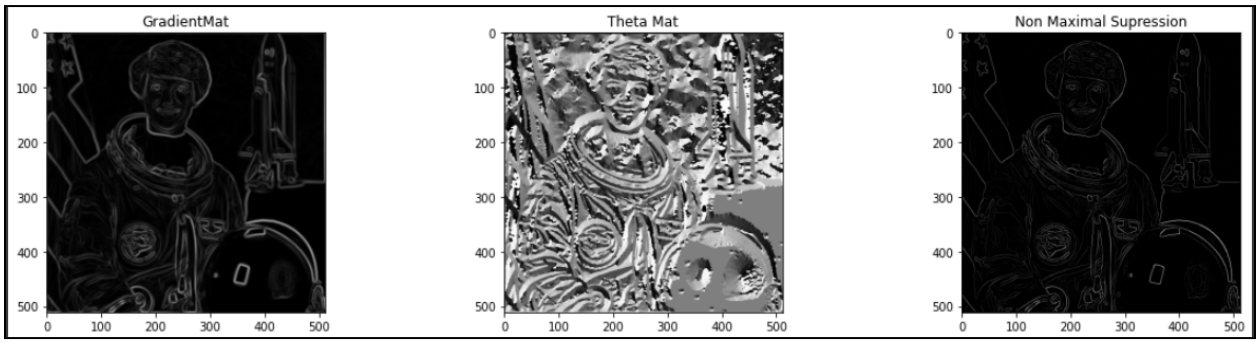


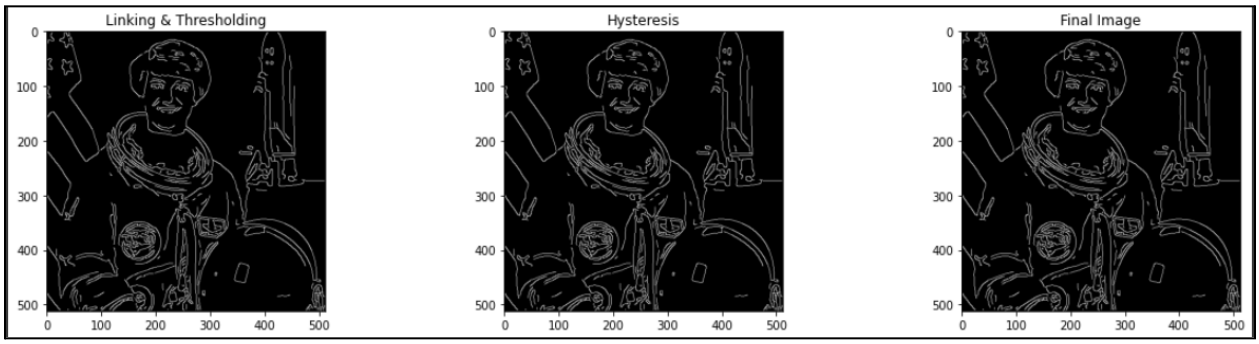


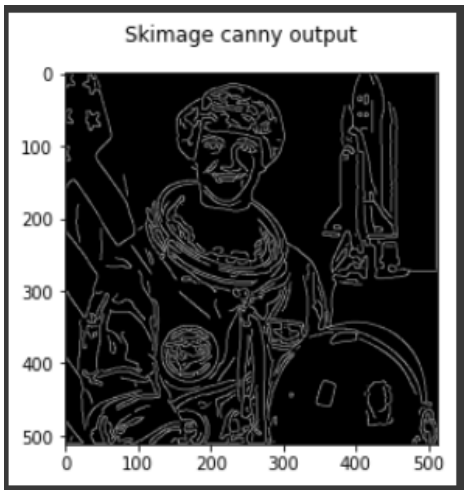


PSNR : 14.032288326101426 SSIM: 0.7295087517970784

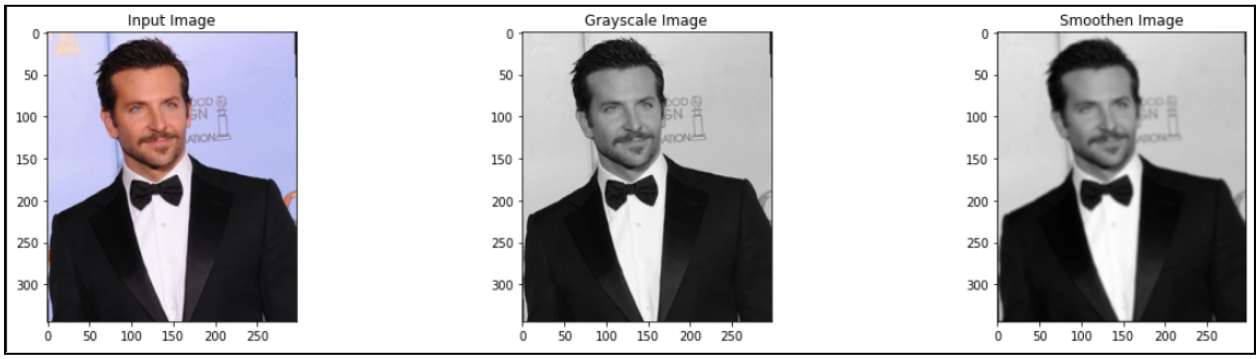


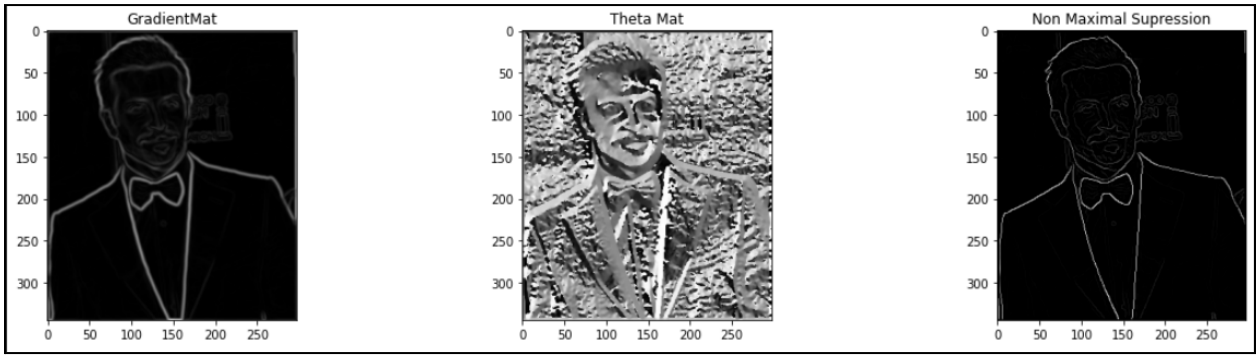


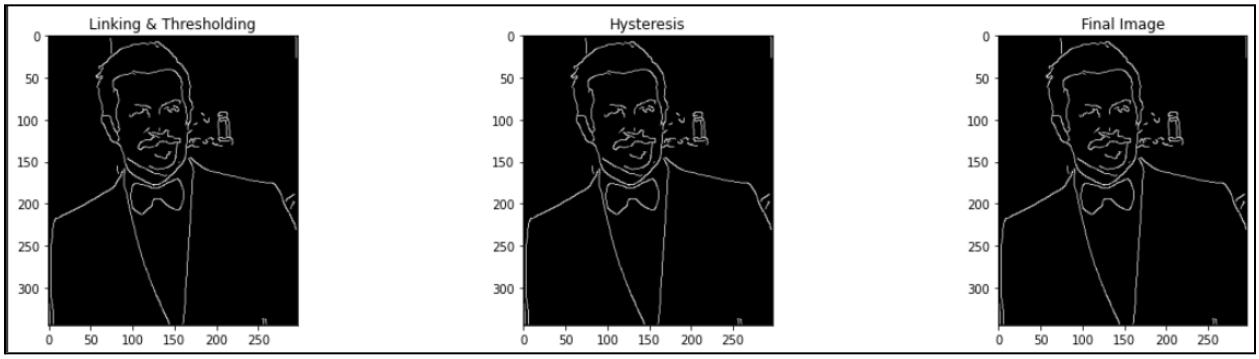


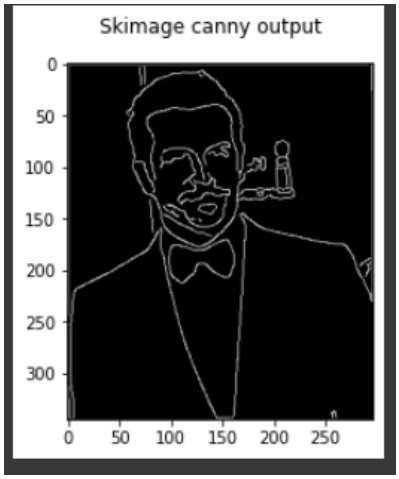


PSNR : 12.615122328435124 SSIM: 0.6531842178510496









PSNR : 17.521831217119793 SSIM: 0.885577508310116

Task-02 Blur Detection

APPROACH:

Whether an image is blurred or not is determined by edges in the image. The most known method for detecting edges from the image is the Laplacian method. As the laplacian operator is used to get the second derivatives of the image. The second derivative of the image can be obtained by the laplacian operator. It basically highlights the regions where there is rapid change of intensity in the image. In this approach the laplacian operator is applied to the image and then calculate the variance of the image pixels.The image with high variance is expected to have sharp edges i.e.it's a clear image, whereas the image with less variance and less maximum are expected to be a blur image.As in process a convolution operation was applied to the image with a Laplacian kernel of size 3x3. After the convolution operation gets completed, variance is calculated.

Observations:

- The threshold value is decided manually on varying different values to get the best possible result.
- As getting the score for image the probability is estimated by

  Maxi = max(score)
  Mini = min(score)
  Normalized_score[i] = (score[i] - mini) / (maxi - mini)

Images : ['2.jpg', '1.jpg', 'blur1.jpeg', '5.jpg', 'blur5.jpg', 'blur4.jpg', '4.jpg', 'blur3.jpeg', 'blur2.jpeg', '6.jpg']

Scores : [0.6957503564661245, 0.5600629306970779, 0.01650542355636891, 2.795814132673293, 0.03472075566811994, 0.14528697429839924, 11.013794266459362, 0.10959377029139716, 0.01139233243343534, 0.4659896462780678]

Probabilities : [0.06220078380487536, 0.04986825618202777, 0.0004647249894698787, 0.25307399392752417, 0.002120302764302705, 0.012169582848167233, 1.0, 0.008925454500463665, 0.0, 0.04131800642900975]

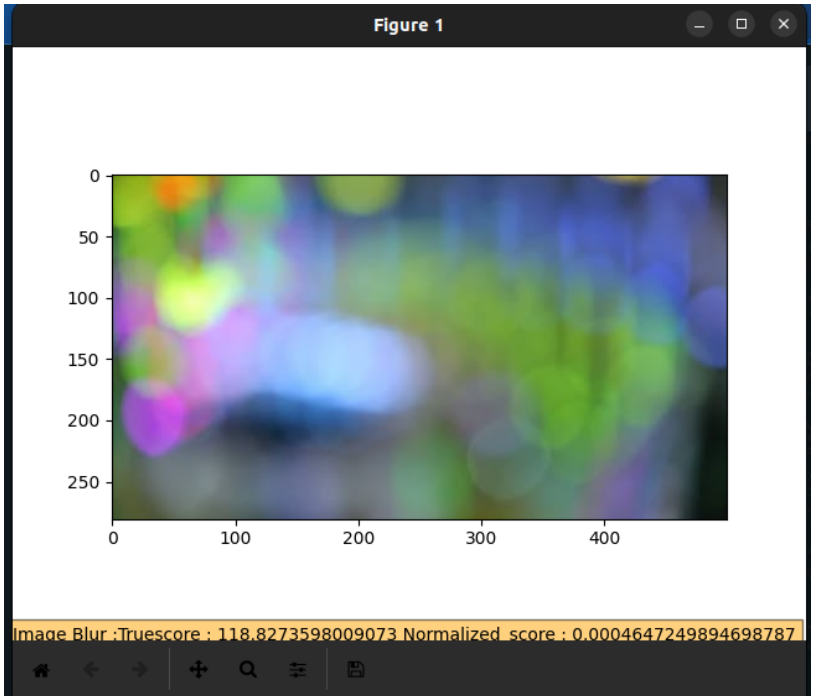


Figure 1
0
50
100
150
200
250
0
100
200
300
400
Image Blur :Truescore : 118.8273598009073 Normalized_score : 0.0004647249894698787

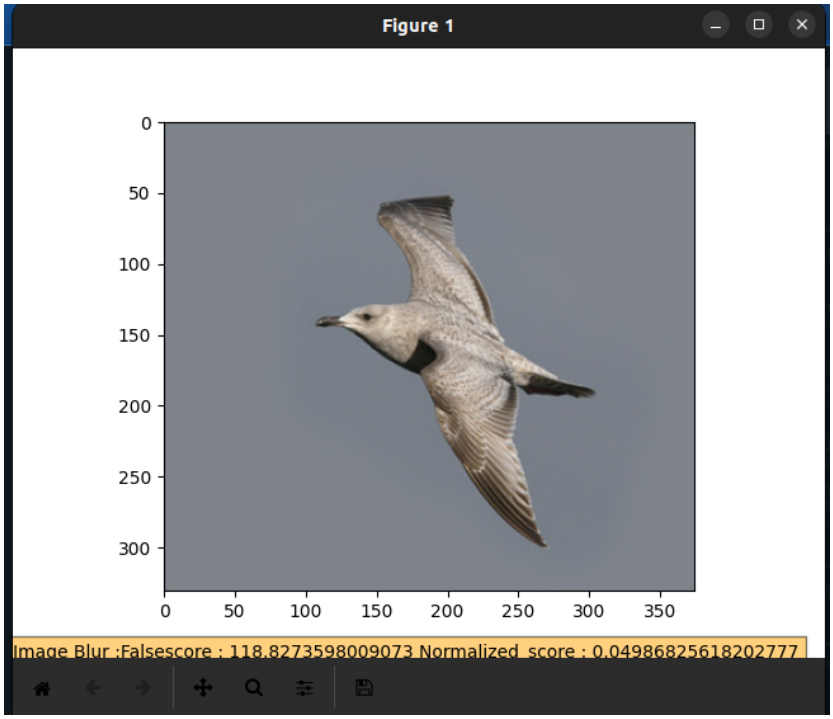


Figure 1
0
50
100
150
200
250
300
0
50
100
150
200
250
300
350
Image Blur :Falsescore : 118.8273598009073 Normalized_score : 0.04986825618202777

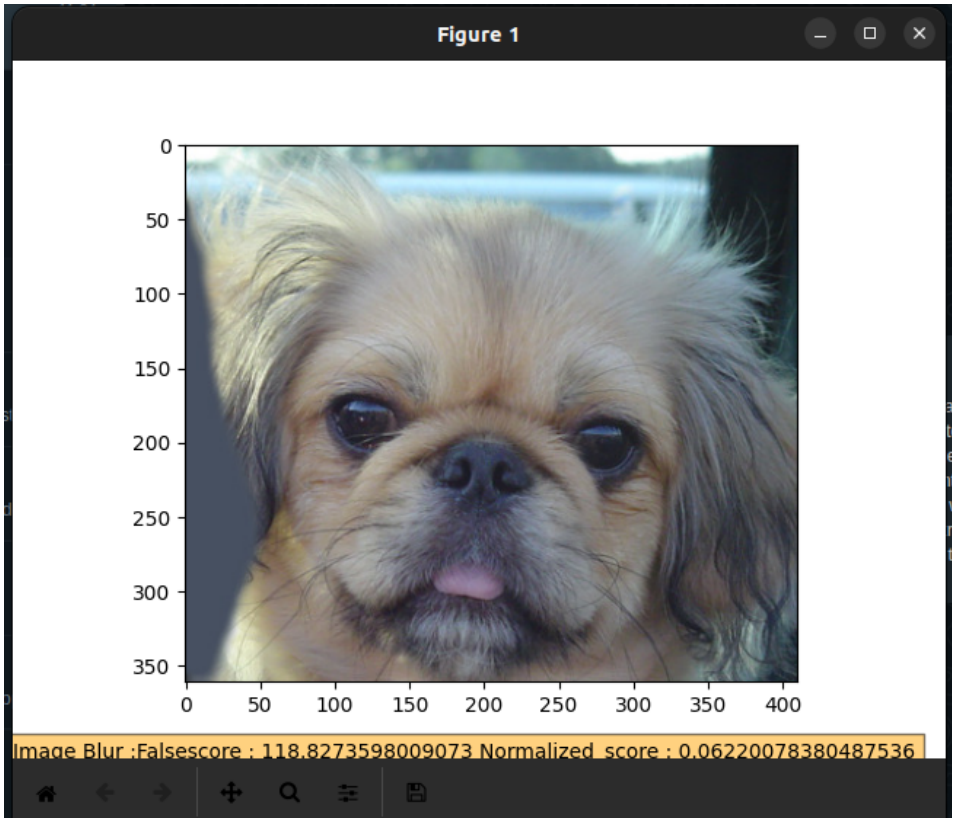


Figure 1
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400
Image Blur :Falsescore : 118.8273598009073 Normalized_score : 0.06220078380487536