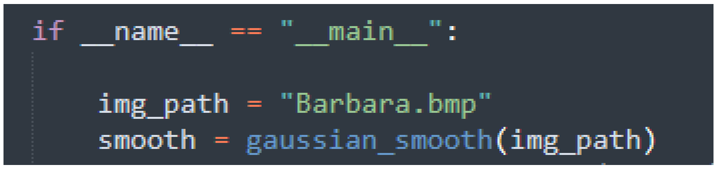
Project 1: Canny’s Edge Detector
Cat Mai

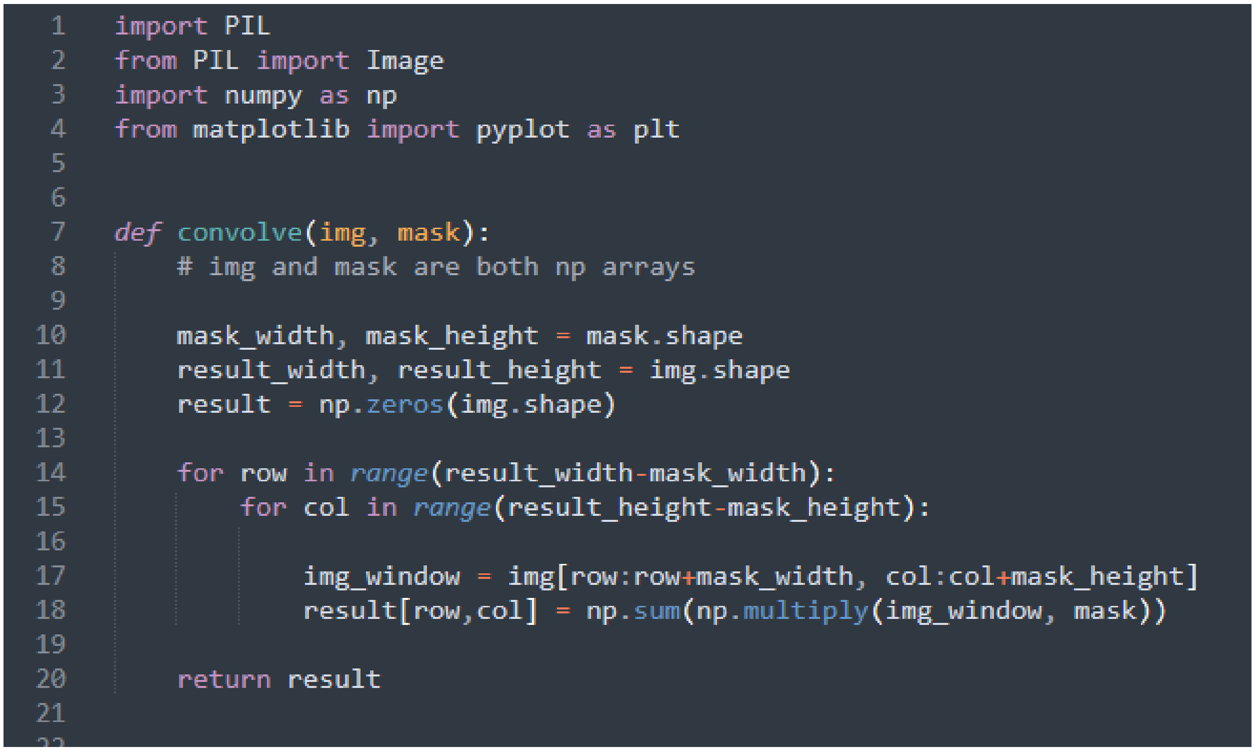
Requirements

1. Being able to compile Python code
2. Install Pillow and numpy, on some command line
    a. pip install Pillow
    b. pip install numpy
3. Change img_path in main.py

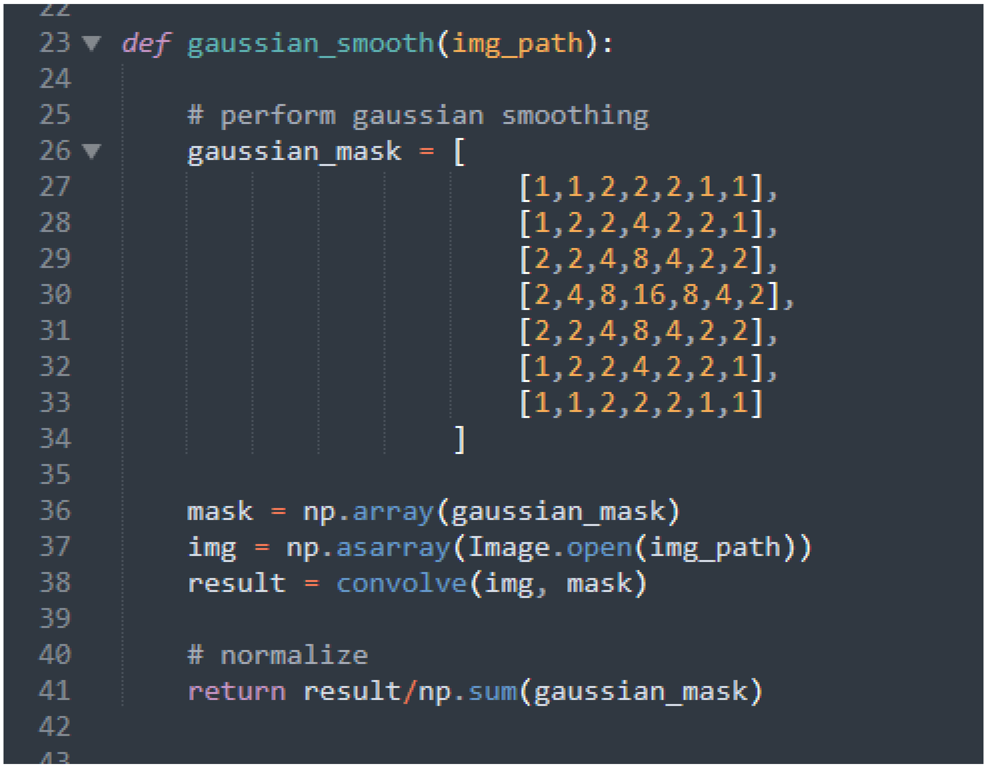


```
if __name__ == "__main__":

    img_path = "Barbara.bmp"
    smooth = gaussian_smooth(img_path)
```

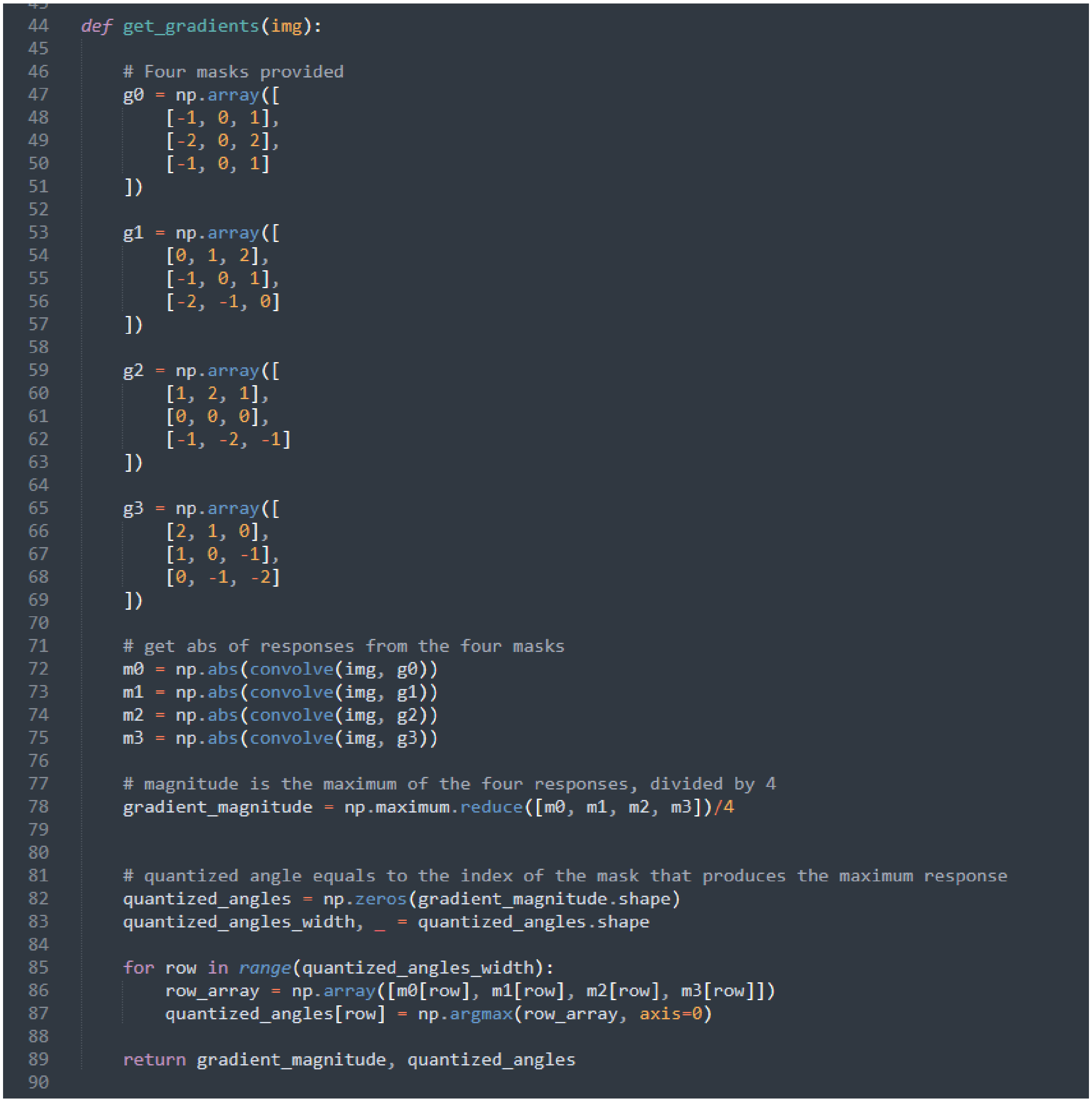
Source Code



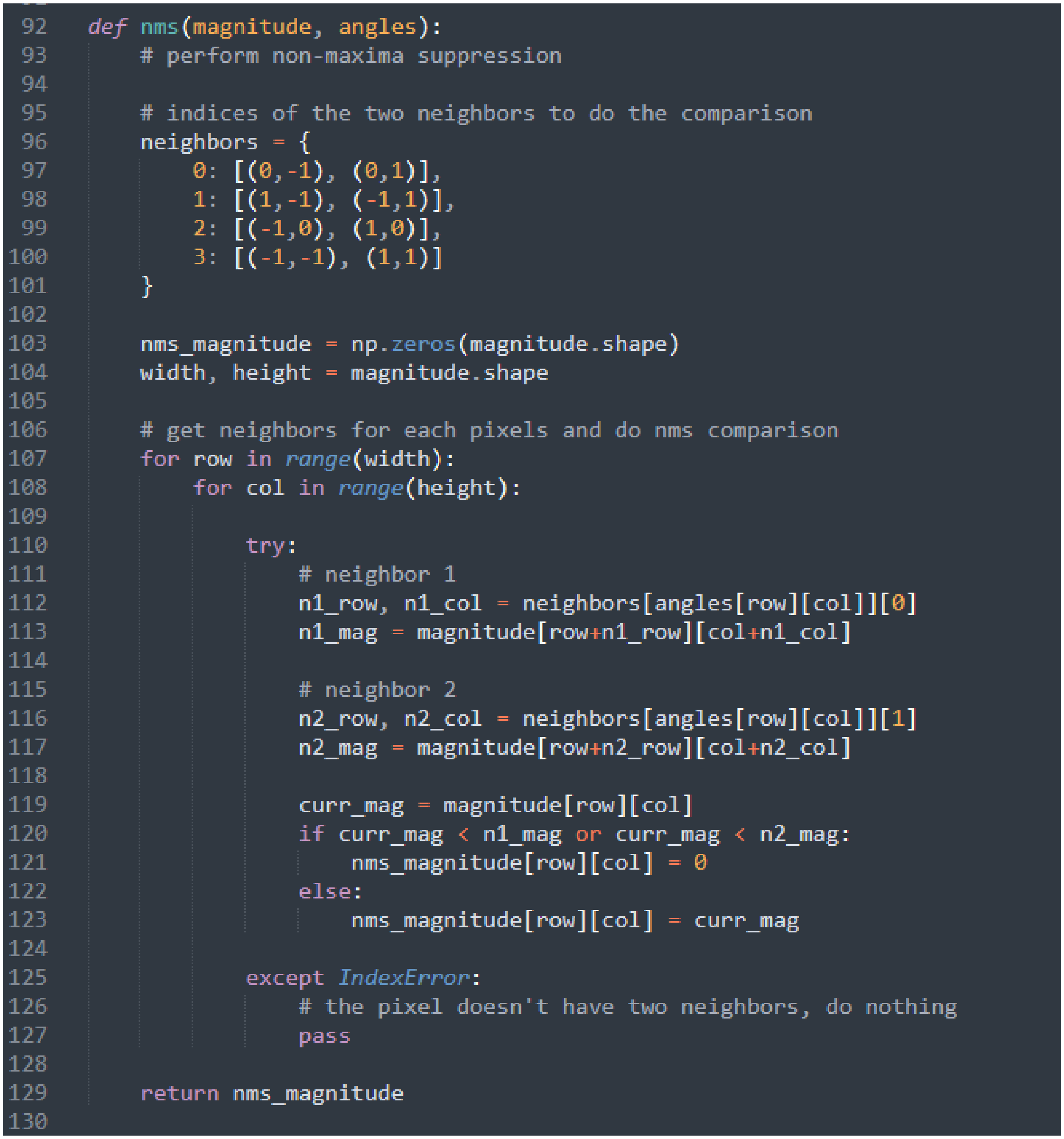
```
import PIL
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt


def convolve(img, mask):
    # img and mask are both np arrays

    mask_width, mask_height = mask.shape
    result_width, result_height = img.shape
    result = np.zeros(img.shape)

    for row in range(result_width-mask_width):
        for col in range(result_height-mask_height):

            img_window = img[row:row+mask_width, col:col+mask_height]
            result[row,col] = np.sum(np.multiply(img_window, mask))

    return result

```



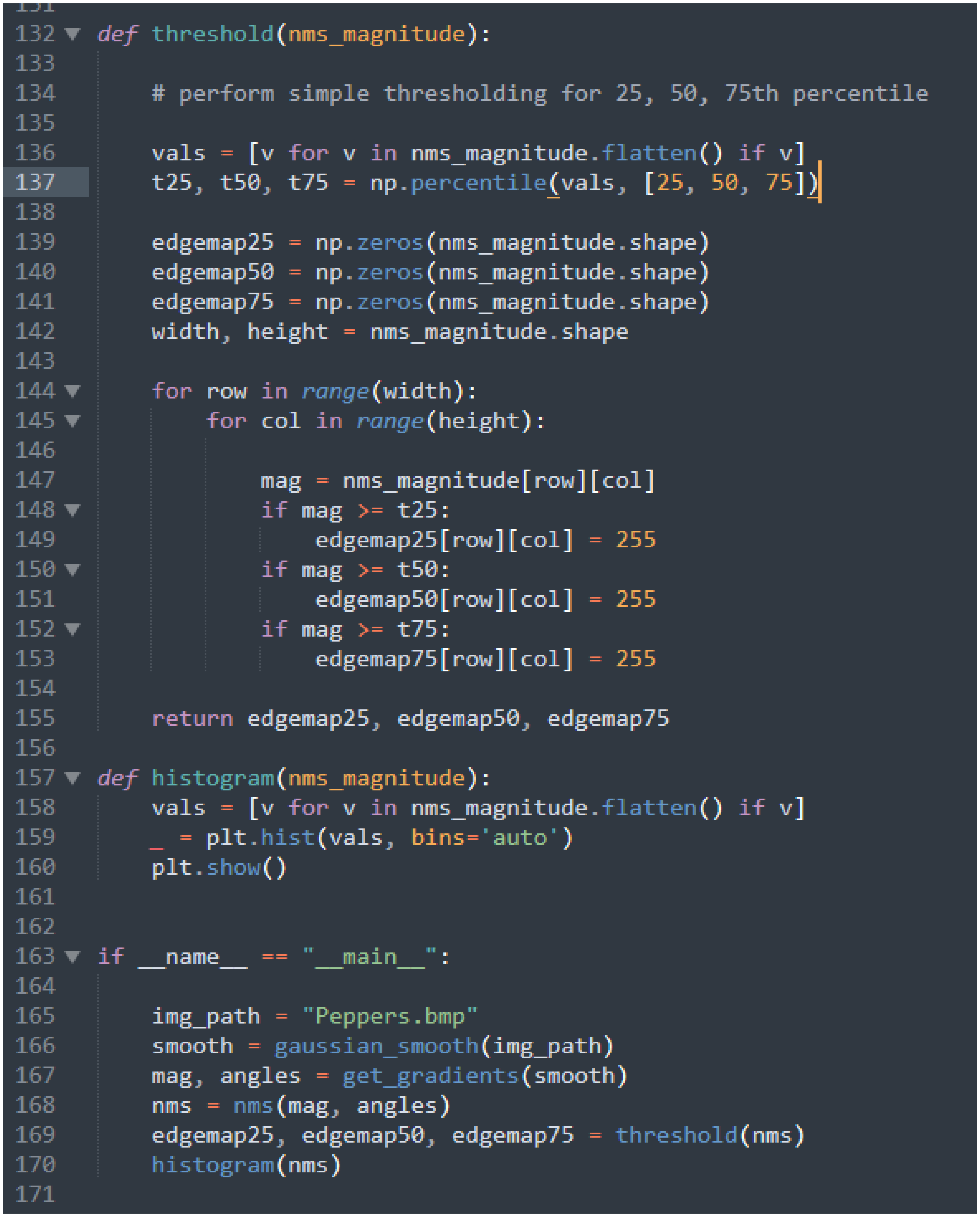
```
def gaussian_smooth(img_path):

    # perform gaussian smoothing
    gaussian_mask = [
                        [1,1,2,2,2,1,1],
                        [1,2,2,4,2,2,1],
                        [2,2,4,8,4,2,2],
                        [2,4,8,16,8,4,2],
                        [2,2,4,8,4,2,2],
                        [1,2,2,4,2,2,1],
                        [1,1,2,2,2,1,1]
                    ]

    mask = np.array(gaussian_mask)
    img = np.asarray(Image.open(img_path))
    result = convolve(img, mask)

    # normalize
    return result/np.sum(gaussian_mask)
```



```python
def get_gradients(img):

    # Four masks provided
    g0 = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ])

    g1 = np.array([
        [0, 1, 2],
        [-1, 0, 1],
        [-2, -1, 0]
    ])

    g2 = np.array([
        [1, 2, 1],
        [0, 0, 0],
        [-1, -2, -1]
    ])

    g3 = np.array([
        [2, 1, 0],
        [1, 0, -1],
        [0, -1, -2]
    ])

    # get abs of responses from the four masks
    m0 = np.abs(convolve(img, g0))
    m1 = np.abs(convolve(img, g1))
    m2 = np.abs(convolve(img, g2))
    m3 = np.abs(convolve(img, g3))

    # magnitude is the maximum of the four responses, divided by 4
    gradient_magnitude = np.maximum.reduce([m0, m1, m2, m3])/4


    # quantized angle equals to the index of the mask that produces the maximum response
    quantized_angles = np.zeros(gradient_magnitude.shape)
    quantized_angles_width, _ = quantized_angles.shape

    for row in range(quantized_angles_width):
        row_array = np.array([m0[row], m1[row], m2[row], m3[row]])
        quantized_angles[row] = np.argmax(row_array, axis=0)

    return gradient_magnitude, quantized_angles

```

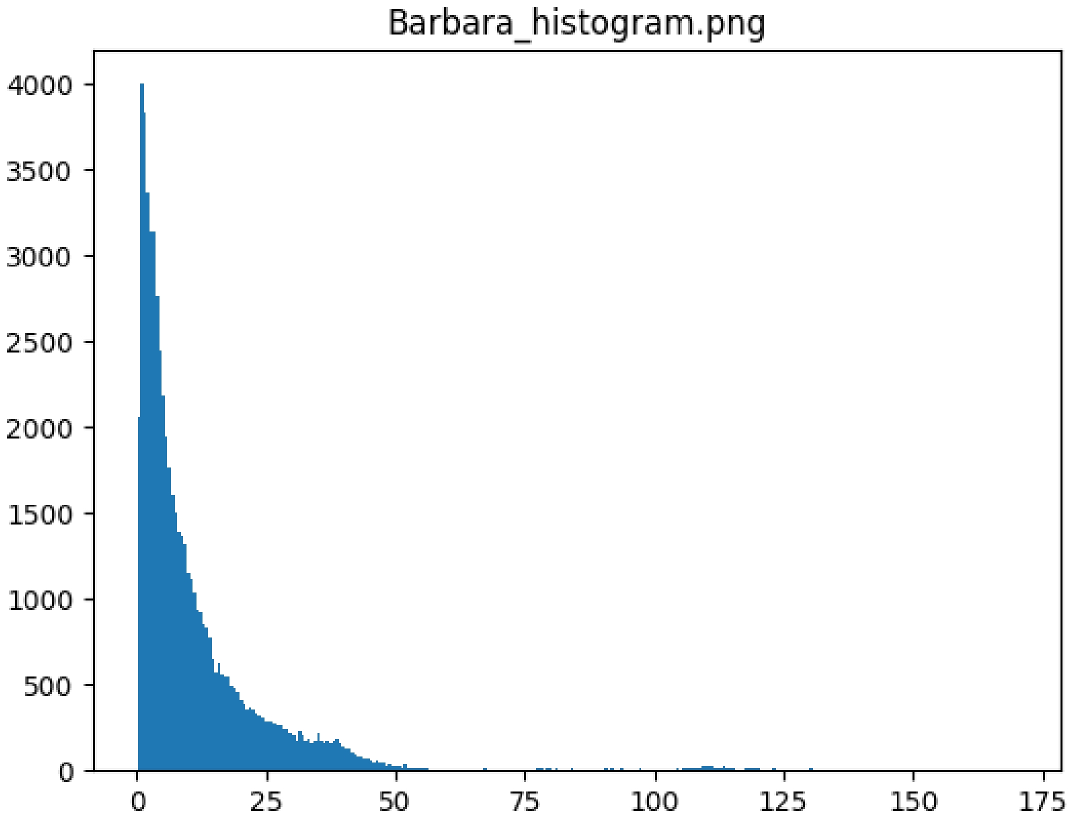


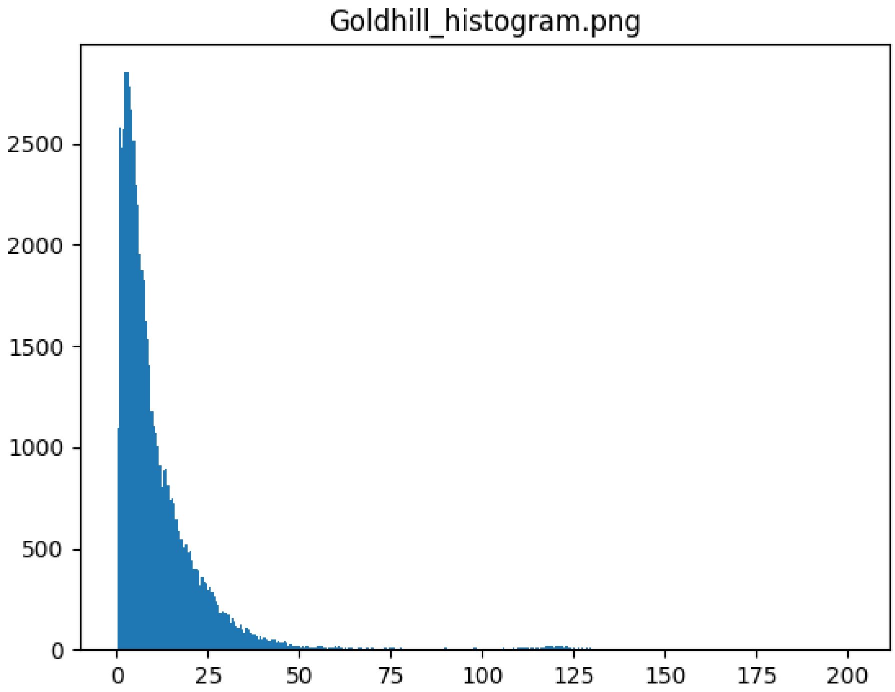
```python
def nms(magnitude, angles):
    # perform non-maxima suppression

    # indices of the two neighbors to do the comparison
    neighbors = {
        0: [(0,-1), (0,1)],
        1: [(1,-1), (-1,1)],
        2: [(-1,0), (1,0)],
        3: [(-1,-1), (1,1)]
    }

    nms_magnitude = np.zeros(magnitude.shape)
    width, height = magnitude.shape

    # get neighbors for each pixels and do nms comparison
    for row in range(width):
        for col in range(height):

            try:
                # neighbor 1
                n1_row, n1_col = neighbors[angles[row][col]][0]
                n1_mag = magnitude[row+n1_row][col+n1_col]

                # neighbor 2
                n2_row, n2_col = neighbors[angles[row][col]][1]
                n2_mag = magnitude[row+n2_row][col+n2_col]

                curr_mag = magnitude[row][col]
                if curr_mag < n1_mag or curr_mag < n2_mag:
                    nms_magnitude[row][col] = 0
                else:
                    nms_magnitude[row][col] = curr_mag

            except IndexError:
                # the pixel doesn't have two neighbors, do nothing
                pass

    return nms_magnitude

```

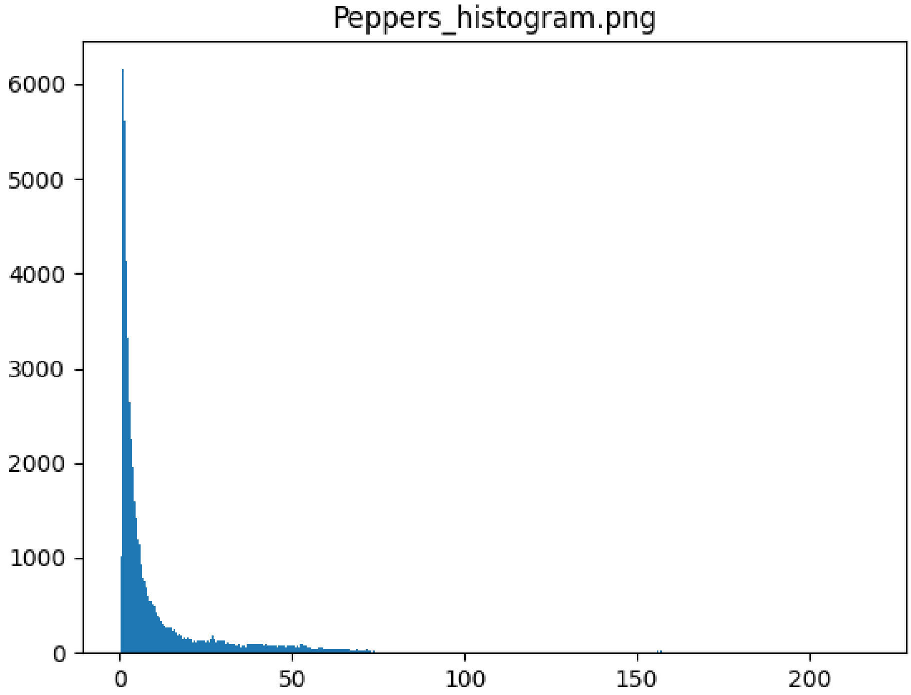


```
def threshold(nms_magnitude):

    # perform simple thresholding for 25, 50, 75th percentile

    vals = [v for v in nms_magnitude.flatten() if v]
    t25, t50, t75 = np.percentile(vals, [25, 50, 75])

    edgemap25 = np.zeros(nms_magnitude.shape)
    edgemap50 = np.zeros(nms_magnitude.shape)
    edgemap75 = np.zeros(nms_magnitude.shape)
    width, height = nms_magnitude.shape

    for row in range(width):
        for col in range(height):

            mag = nms_magnitude[row][col]
            if mag >= t25:
                edgemap25[row][col] = 255
            if mag >= t50:
                edgemap50[row][col] = 255
            if mag >= t75:
                edgemap75[row][col] = 255

    return edgemap25, edgemap50, edgemap75

def histogram(nms_magnitude):
    vals = [v for v in nms_magnitude.flatten() if v]
    _ = plt.hist(vals, bins='auto')
    plt.show()


if __name__ == "__main__":

    img_path = "Peppers.bmp"
    smooth = gaussian_smooth(img_path)
    mag, angles = get_gradients(smooth)
    nms = nms(mag, angles)
    edgemap25, edgemap50, edgemap75 = threshold(nms)
    histogram(nms)
```

Histograms

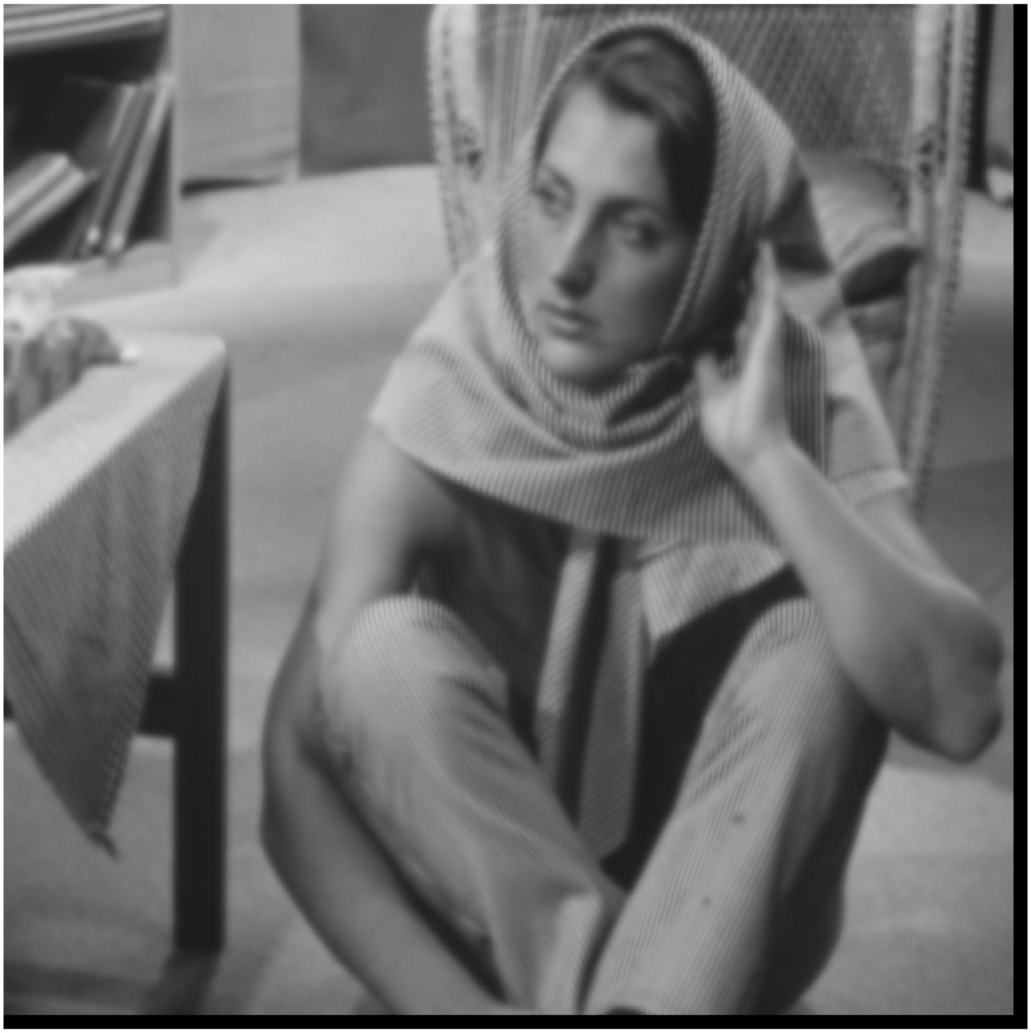

Barbara_histogram.png
4000
3500
3000
2500
2000
1500
1000
500
0
0
25
50
75
100
125
150
175


Goldhill_histogram.png
0
500
1000
1500
2000
2500
0
25
50
75
100
125
150
175
200

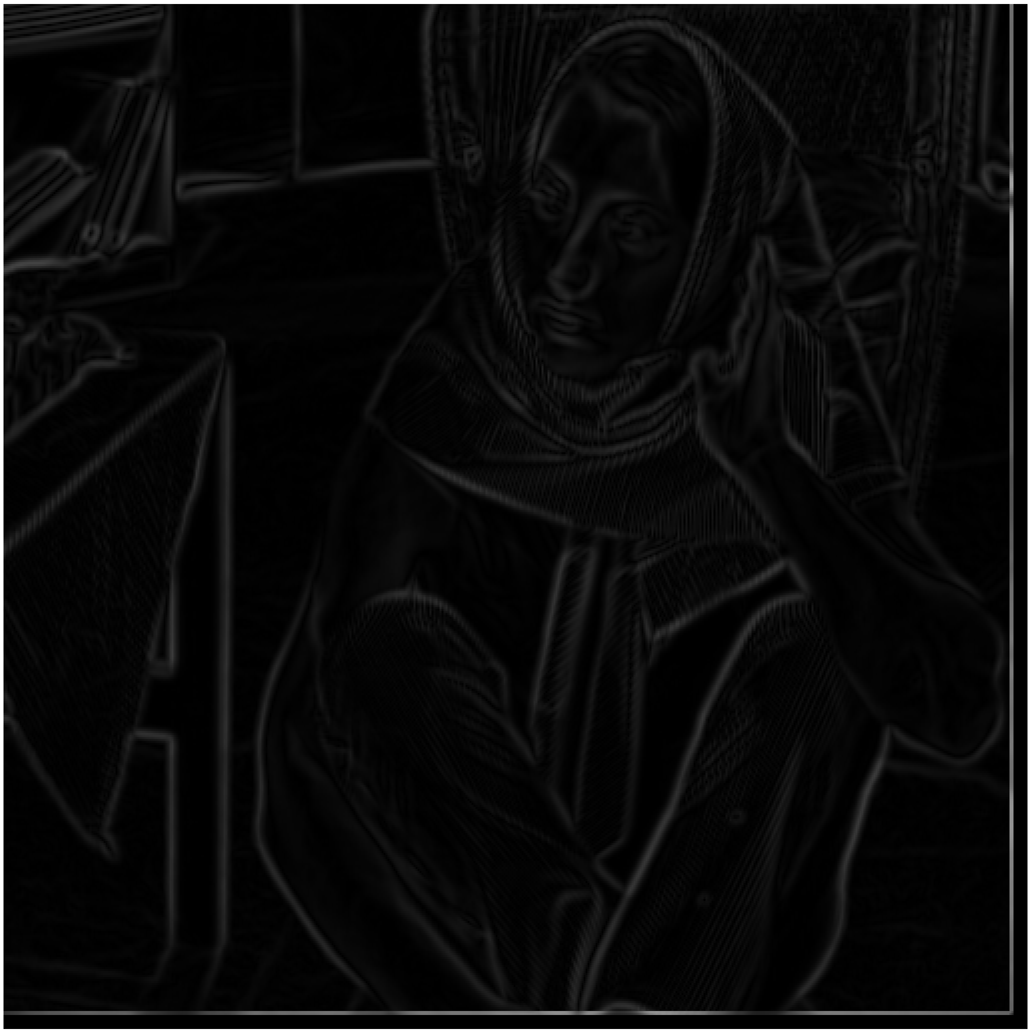

Peppers_histogram.png
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200

Barbara.bmp

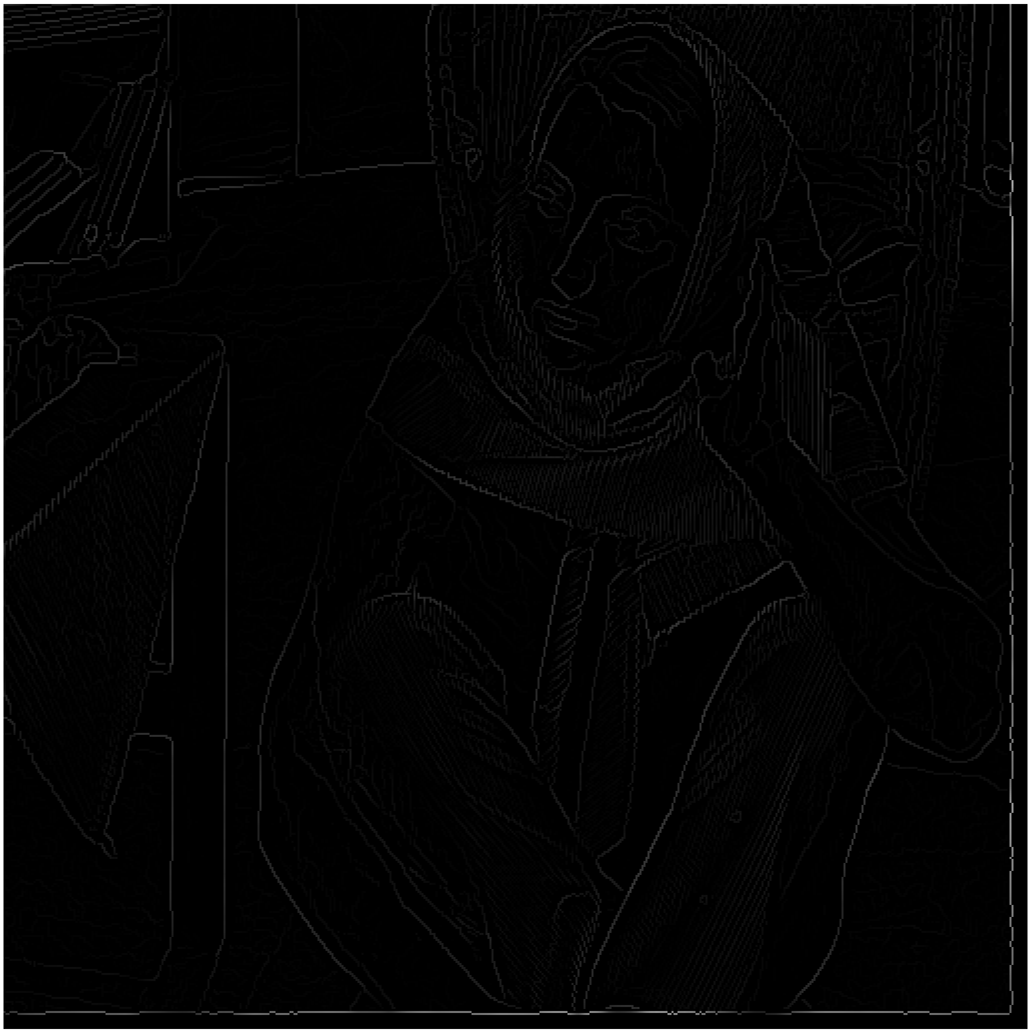
After Gaussian smoothing



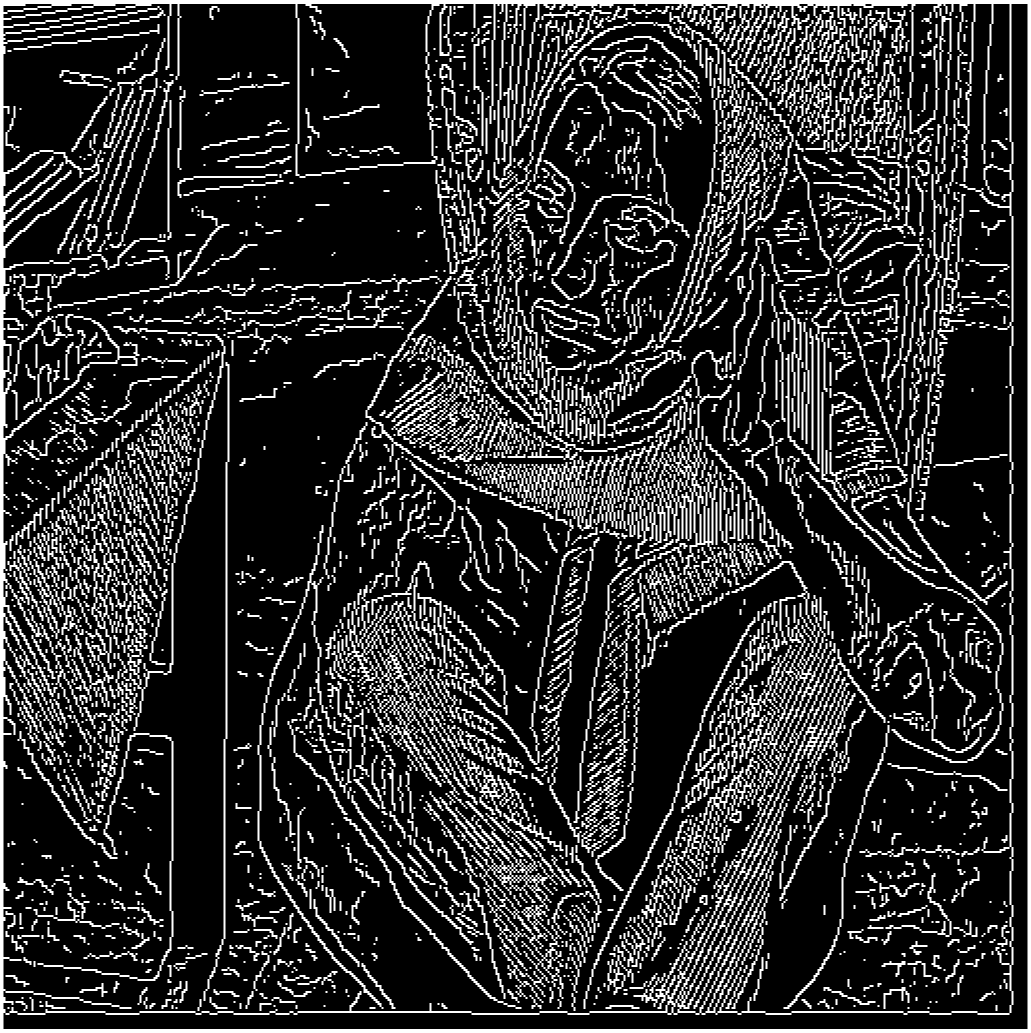
Normalized mag image



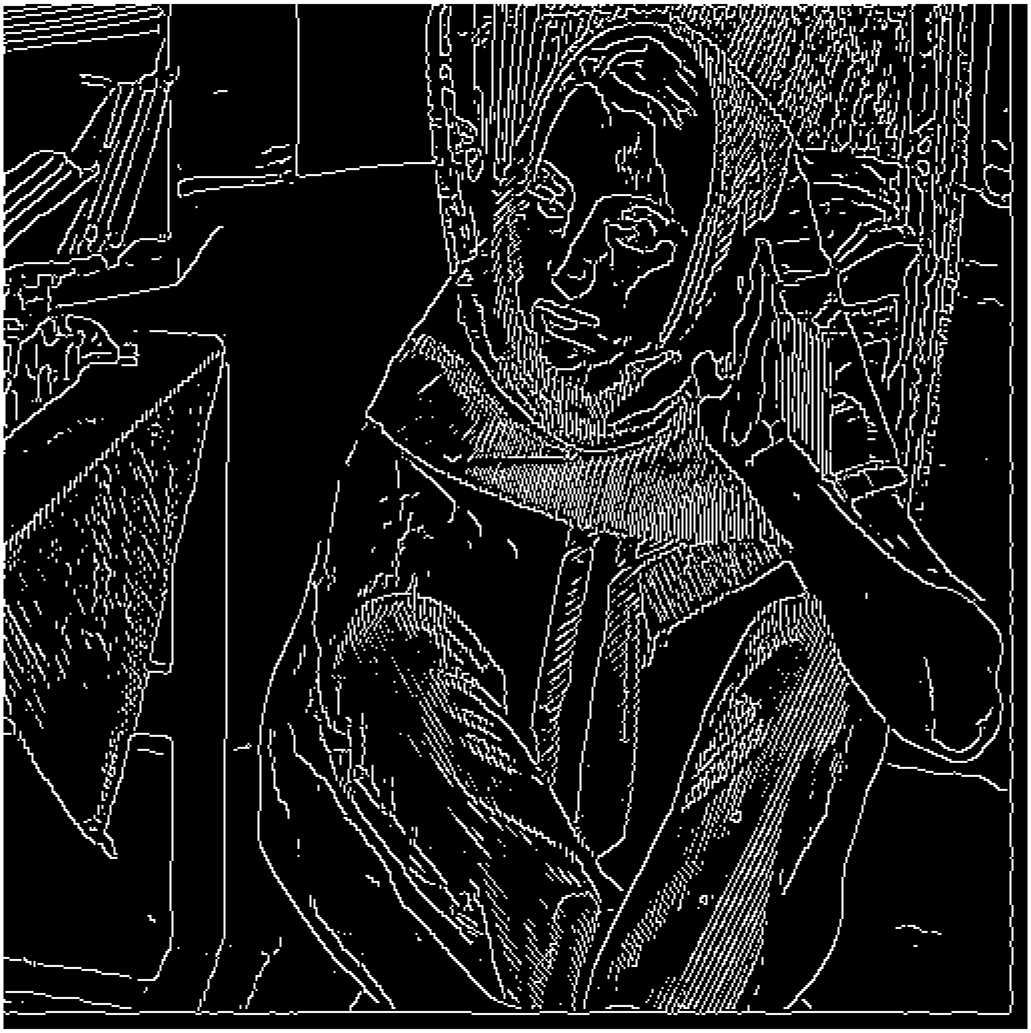
After nms



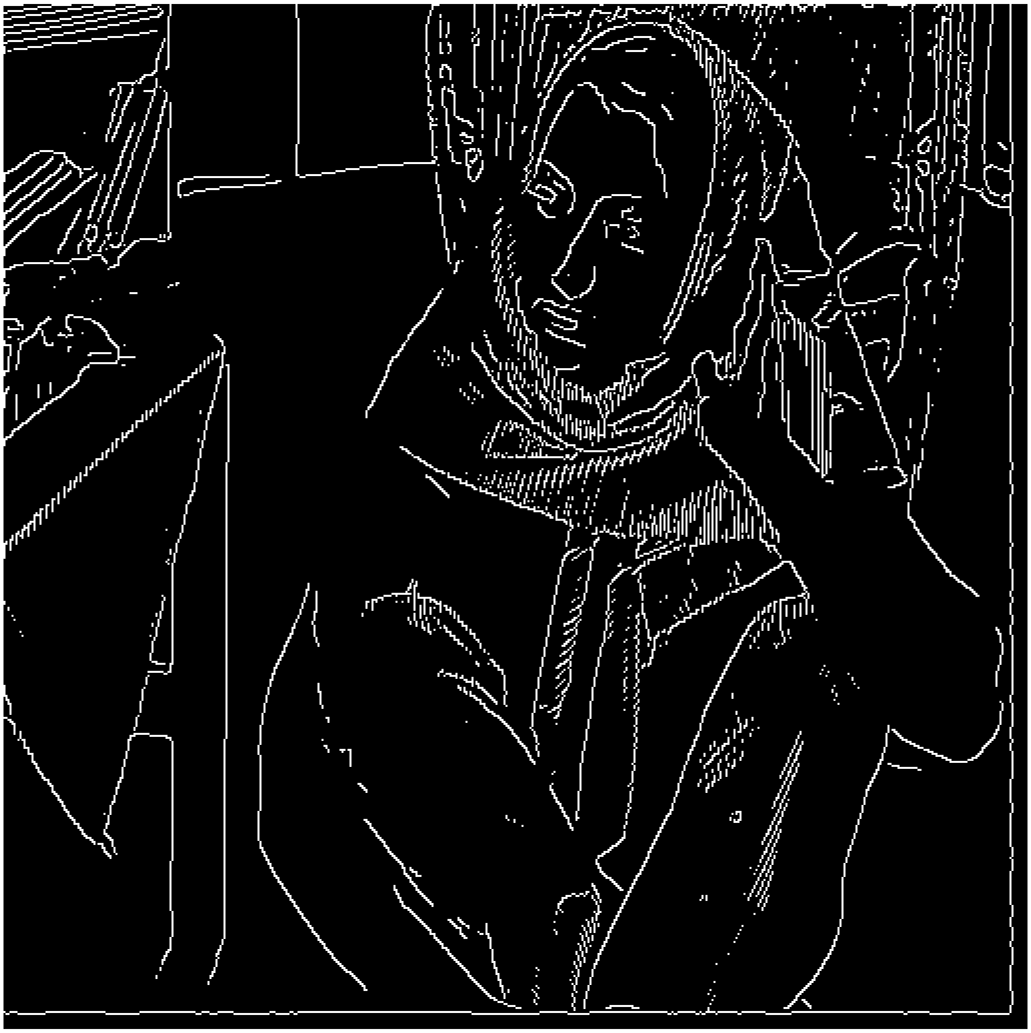
25th Percentile - 2.89



50th Percentile - 6.64

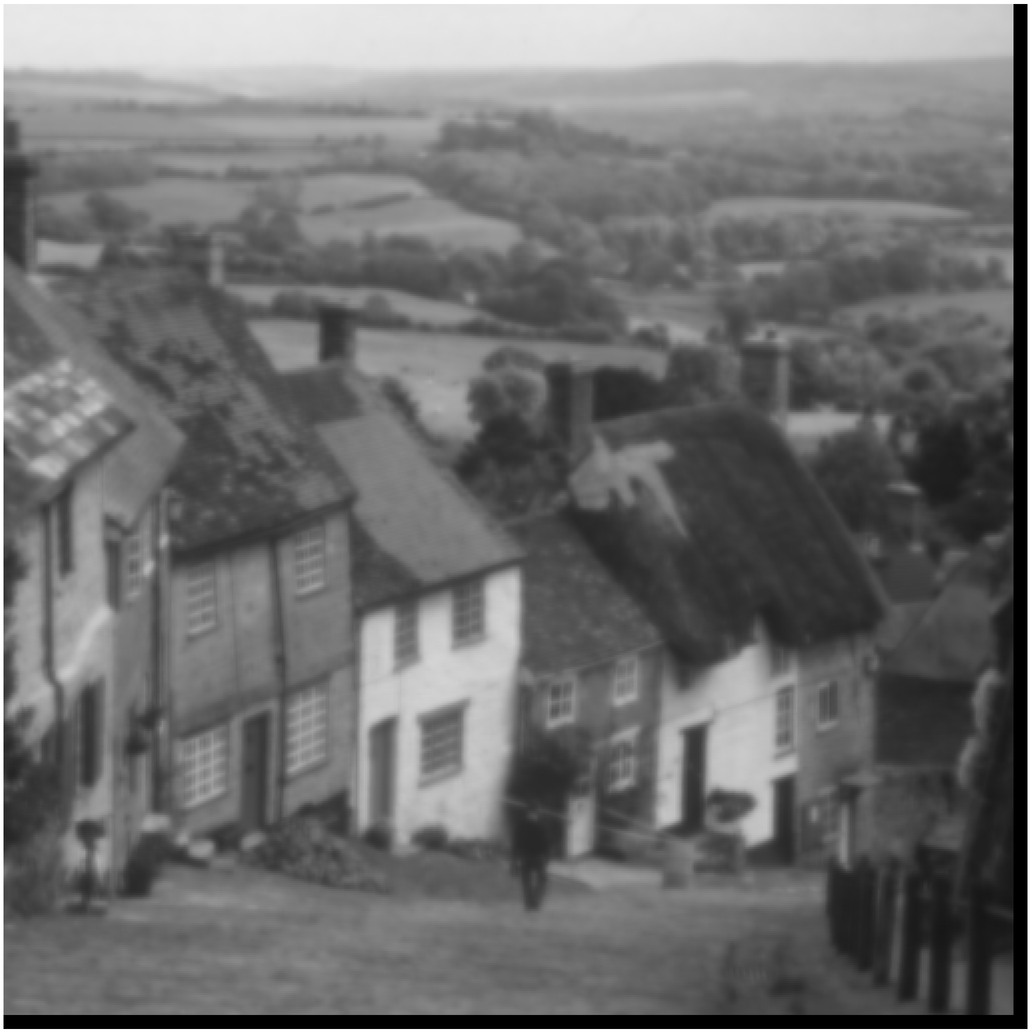


75th Percentile - 14.7

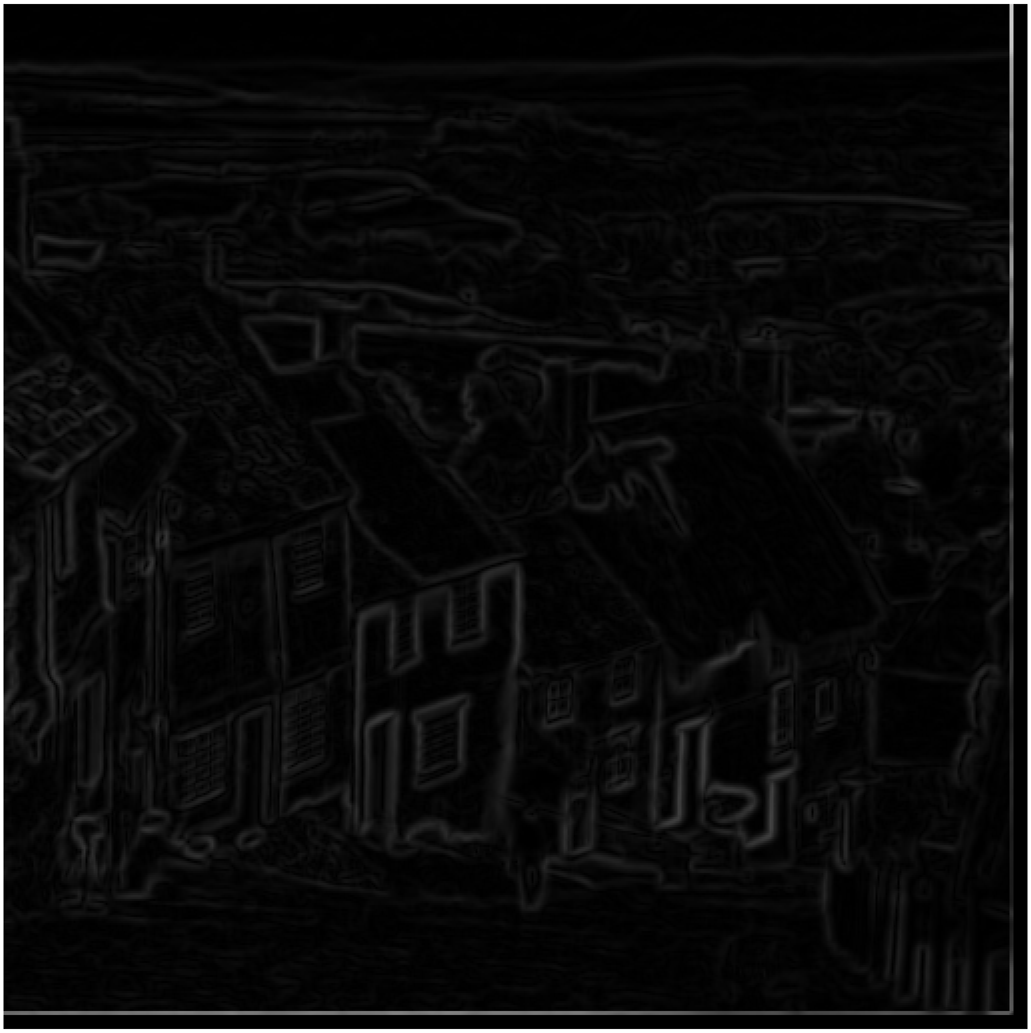


Goldhill.bmp

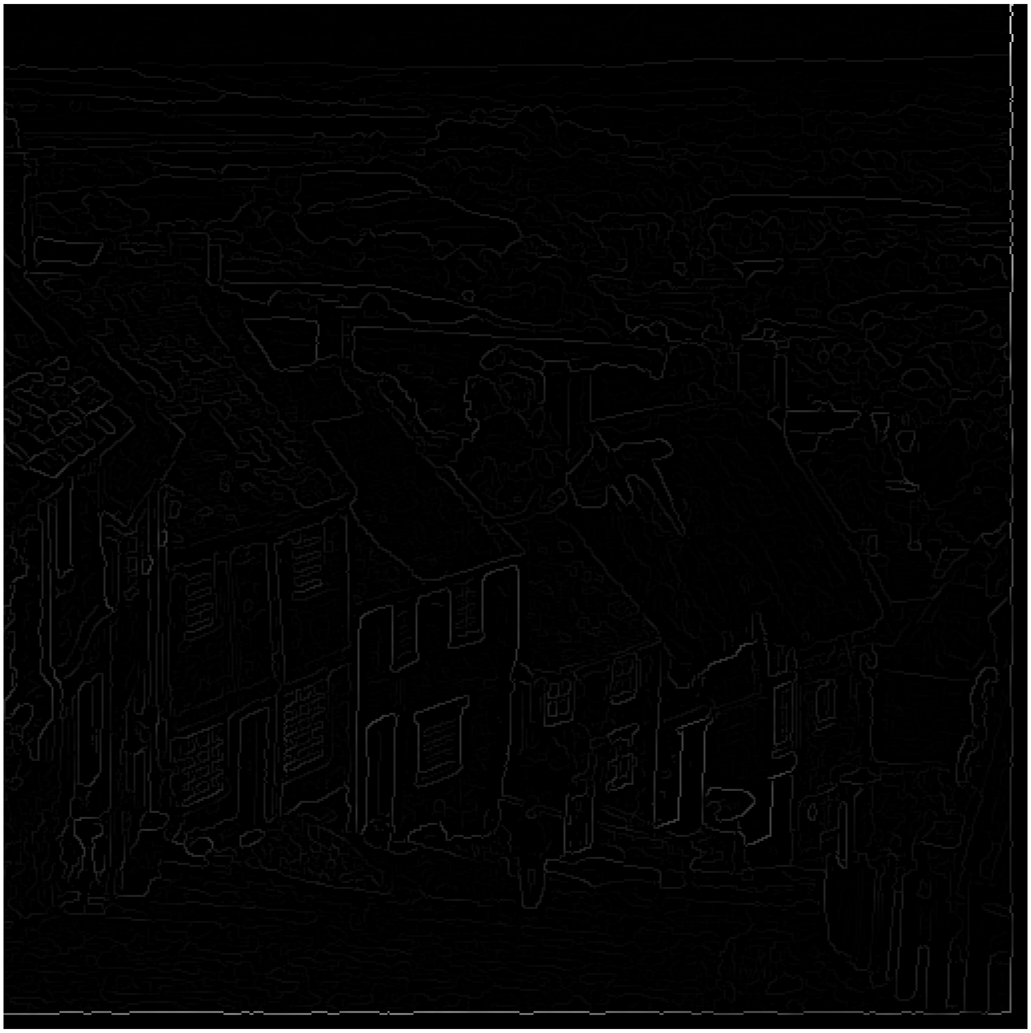
After Gaussian smoothing



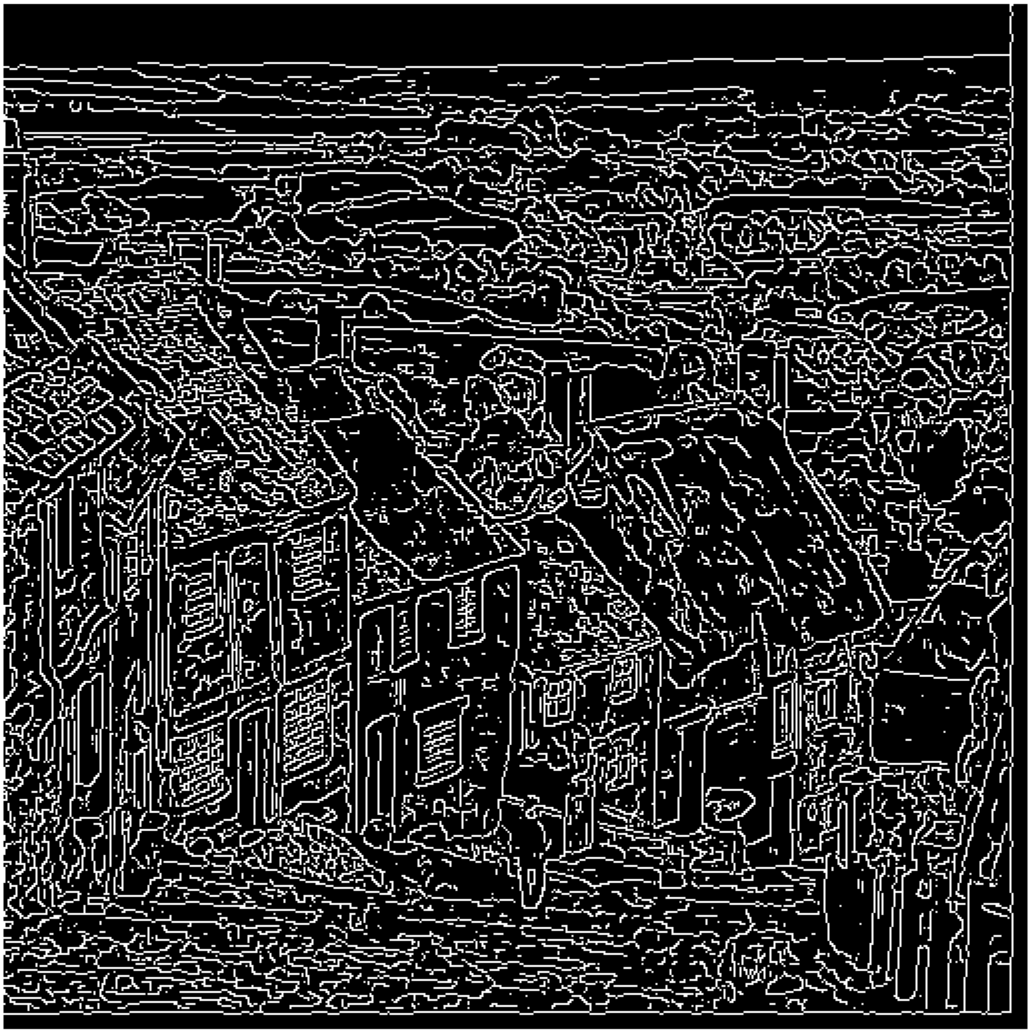
Normalized mag image



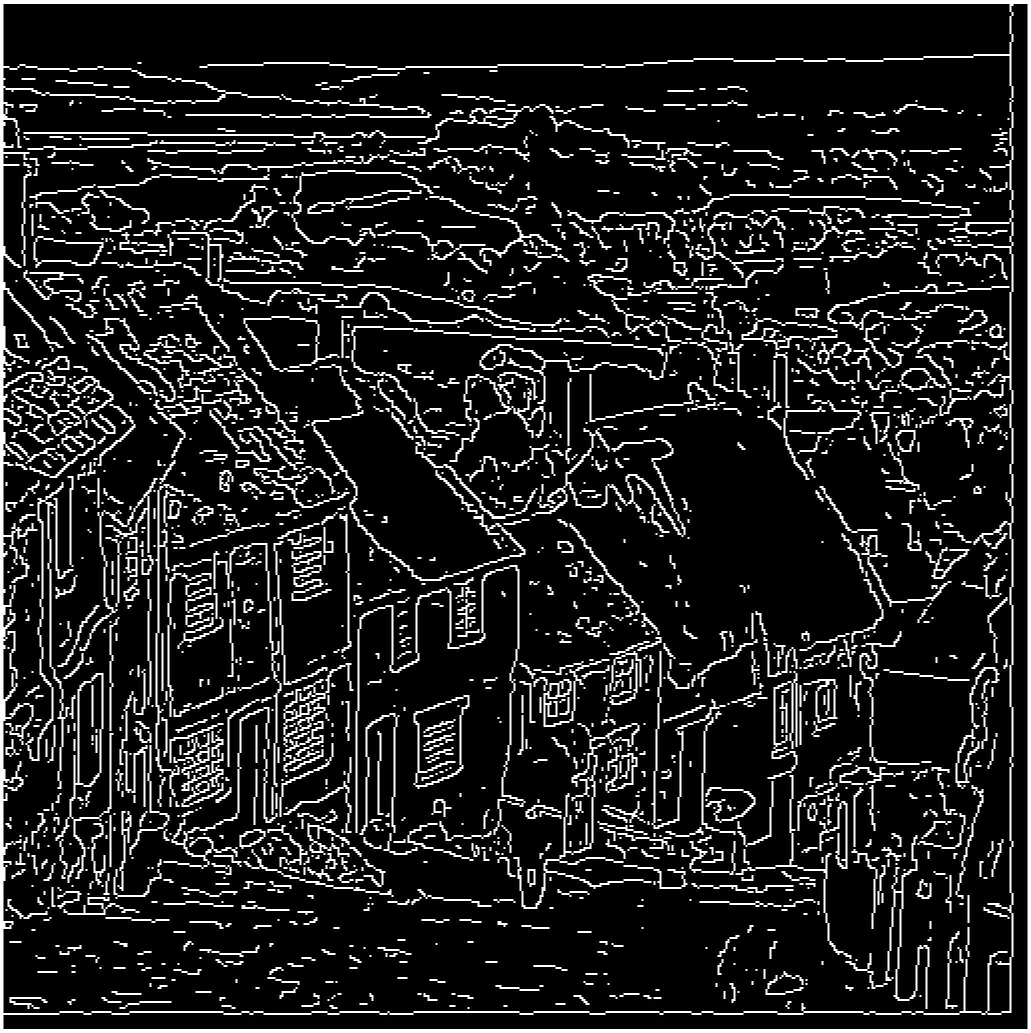
After nms



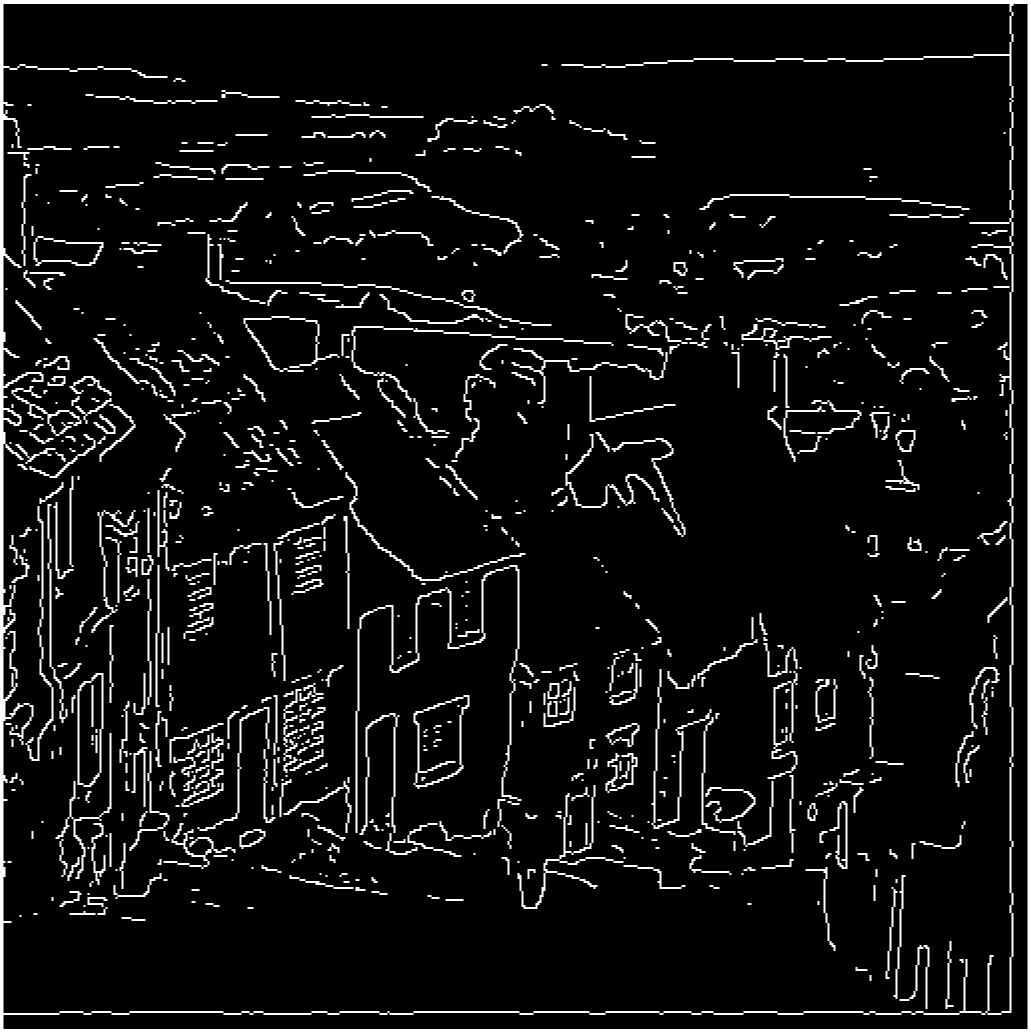
25th Percentile - 3.56



50th Percentile - 7.2

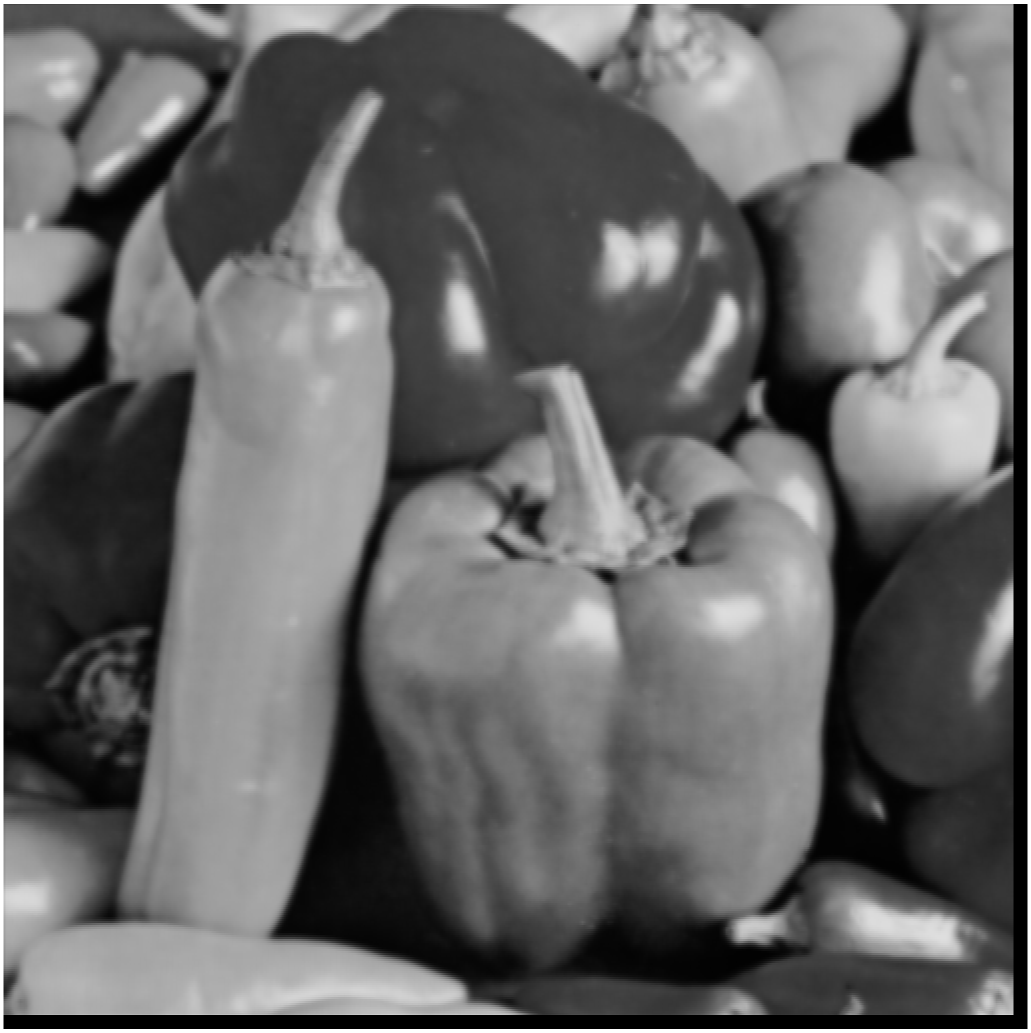


75th Percentile - 14.4

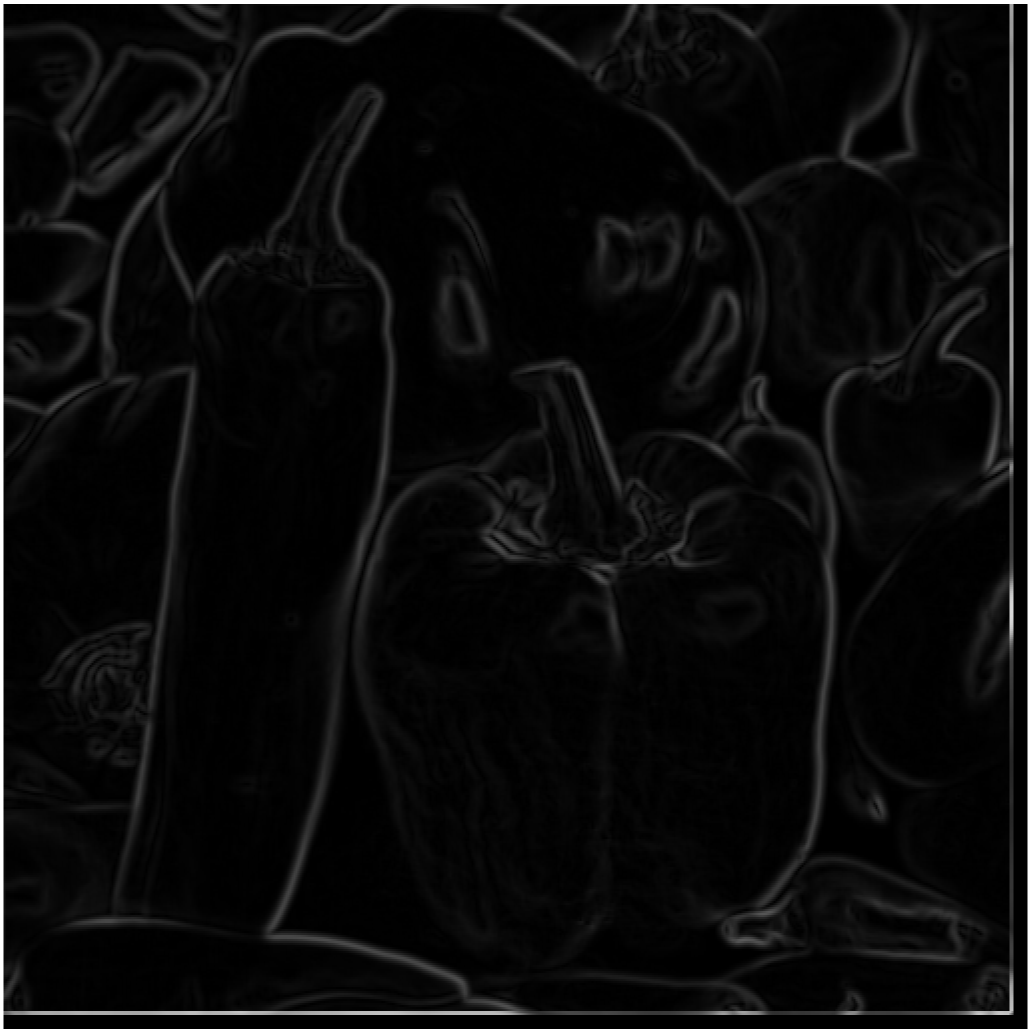


Peppers.bmp

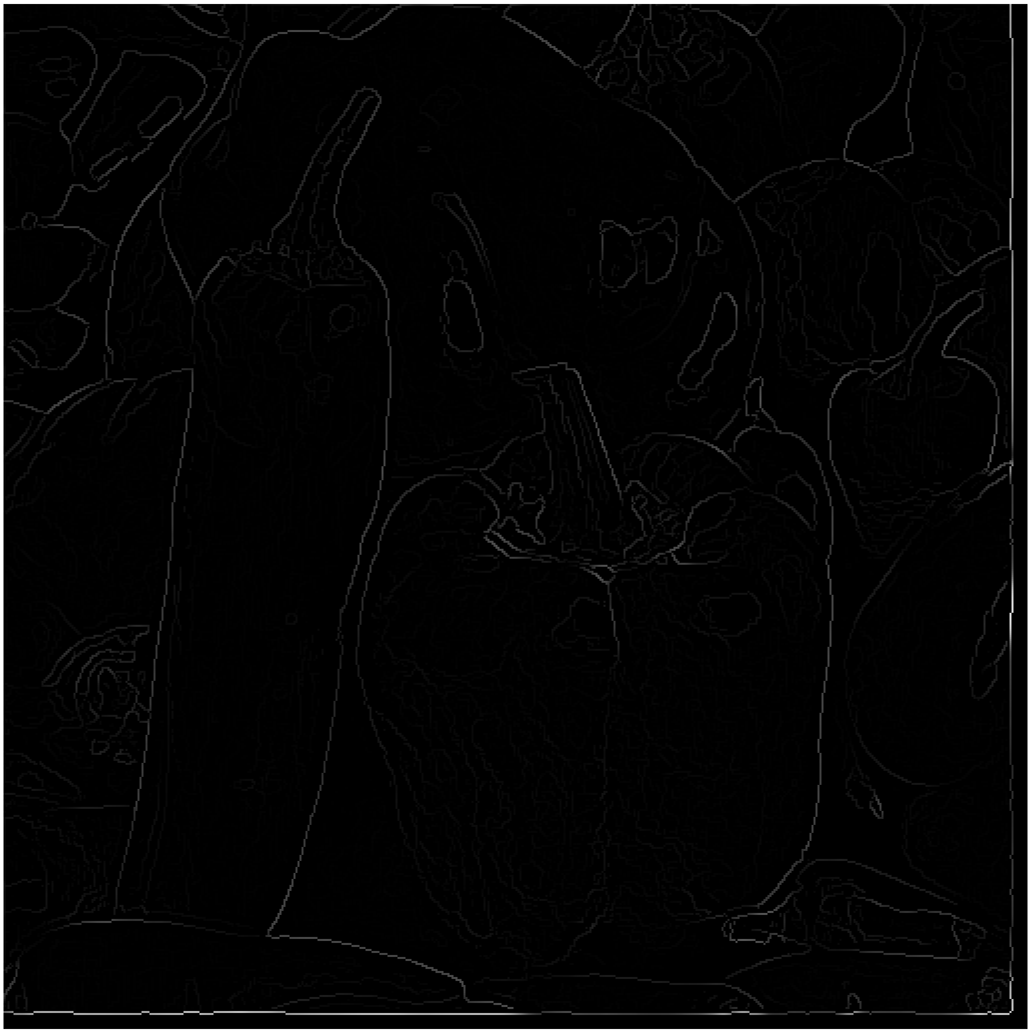
After Gaussian smoothing



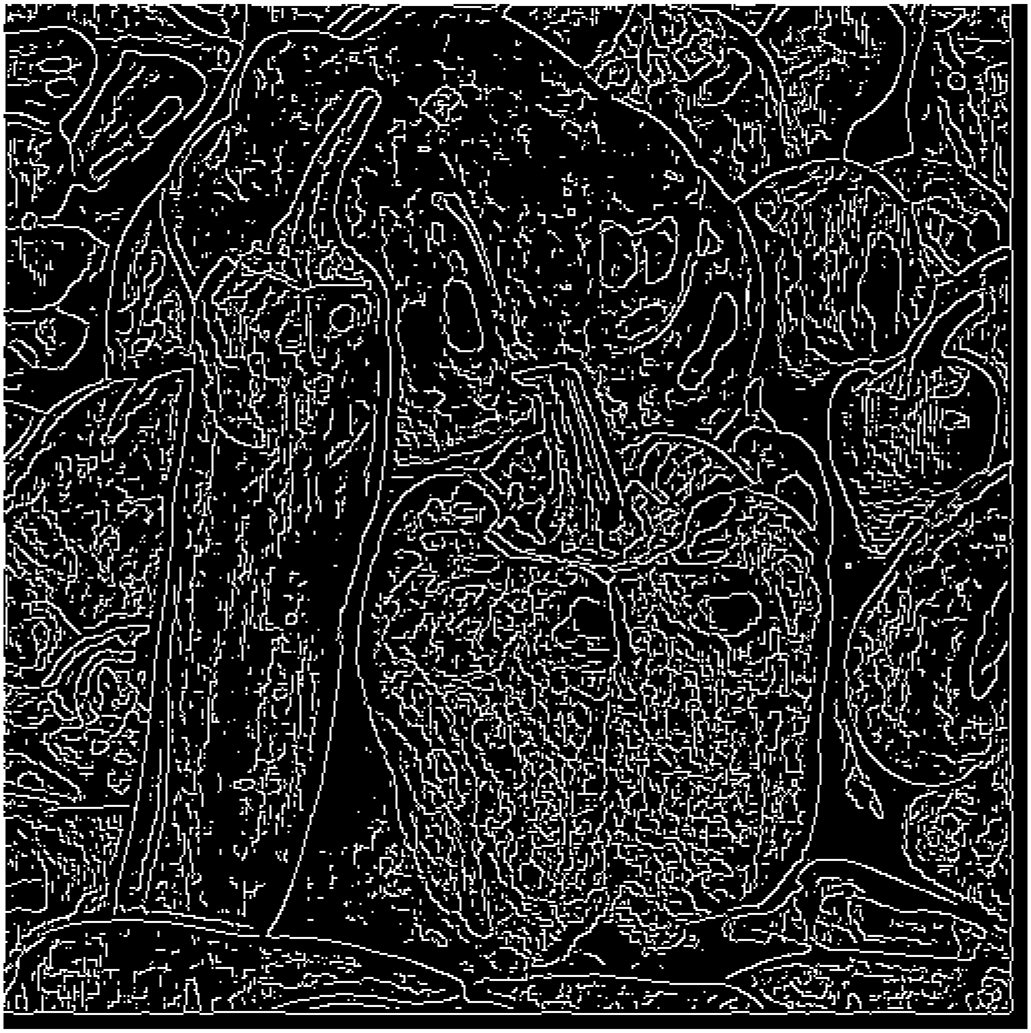
Normalized mag image



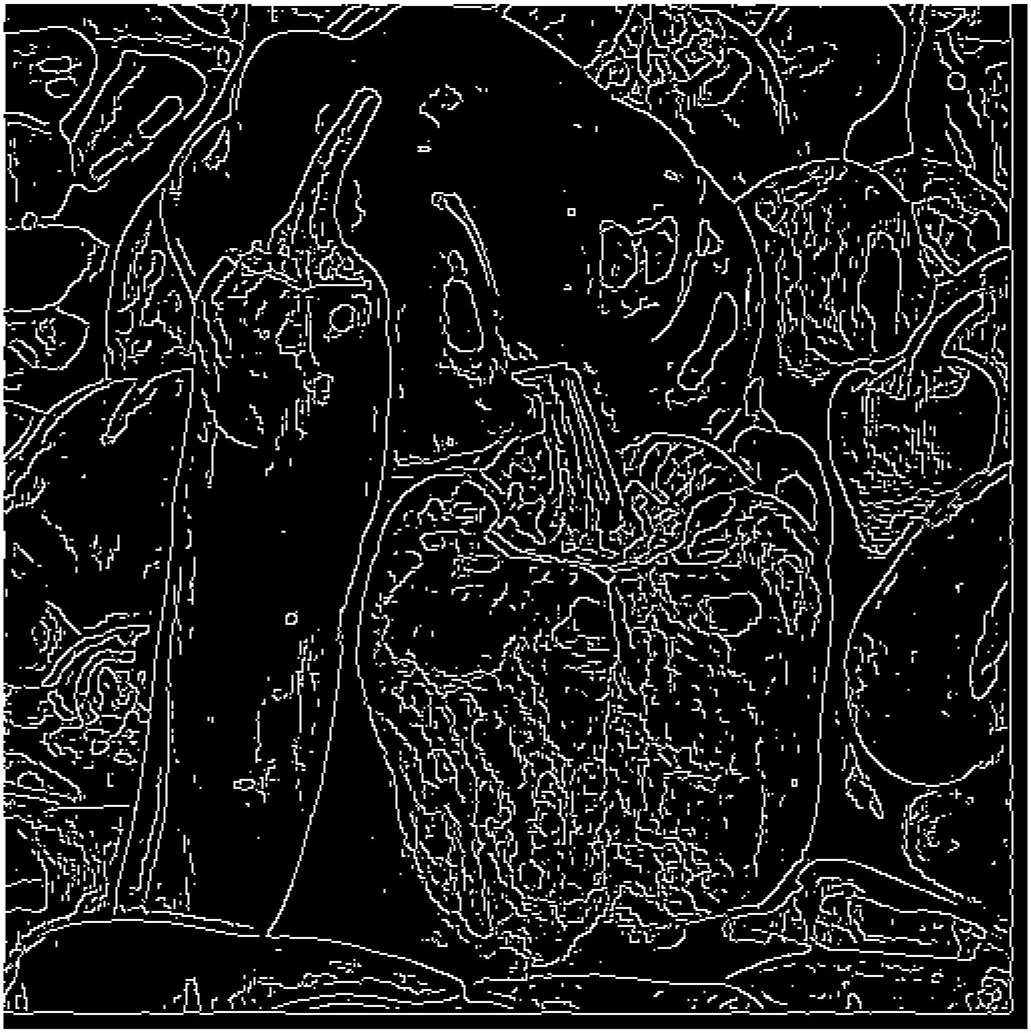
After nms



25th Percentile - 1.69



50th Percentile - 3.59



75th Percentile - 10.56