<u>Computer Vision Project 1</u>

<u>Canny Edge Detector</u>

Student 1 – Harsh Sanjay Apte

NetID – ha2179

Student 2 – Savani Manoj Gokhale

NetID – sg6428

How to run program? –

Run the python file by following command

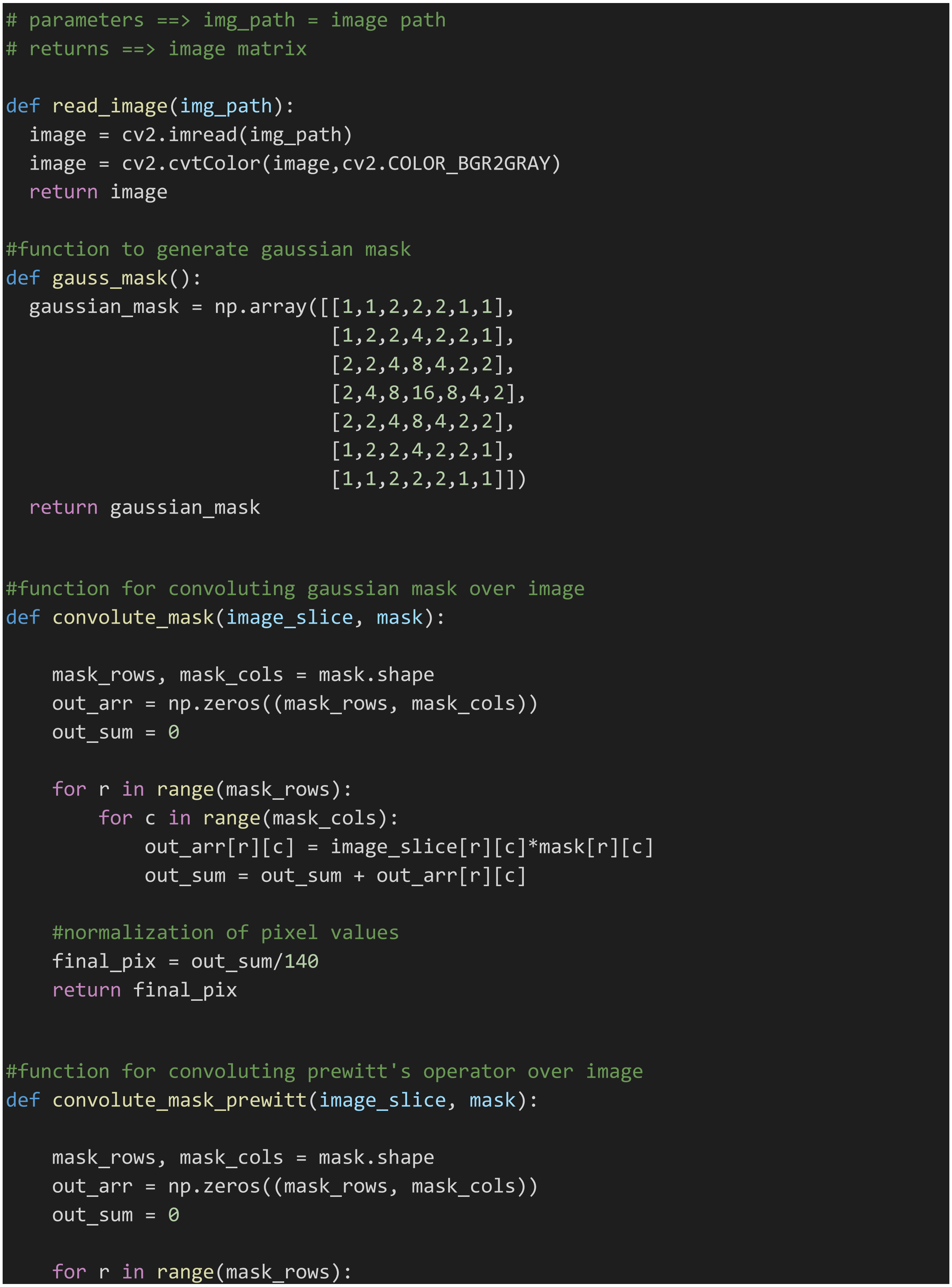
python canny_edge_final.py - -*path = “<path-to-file>”*

Required Packages –

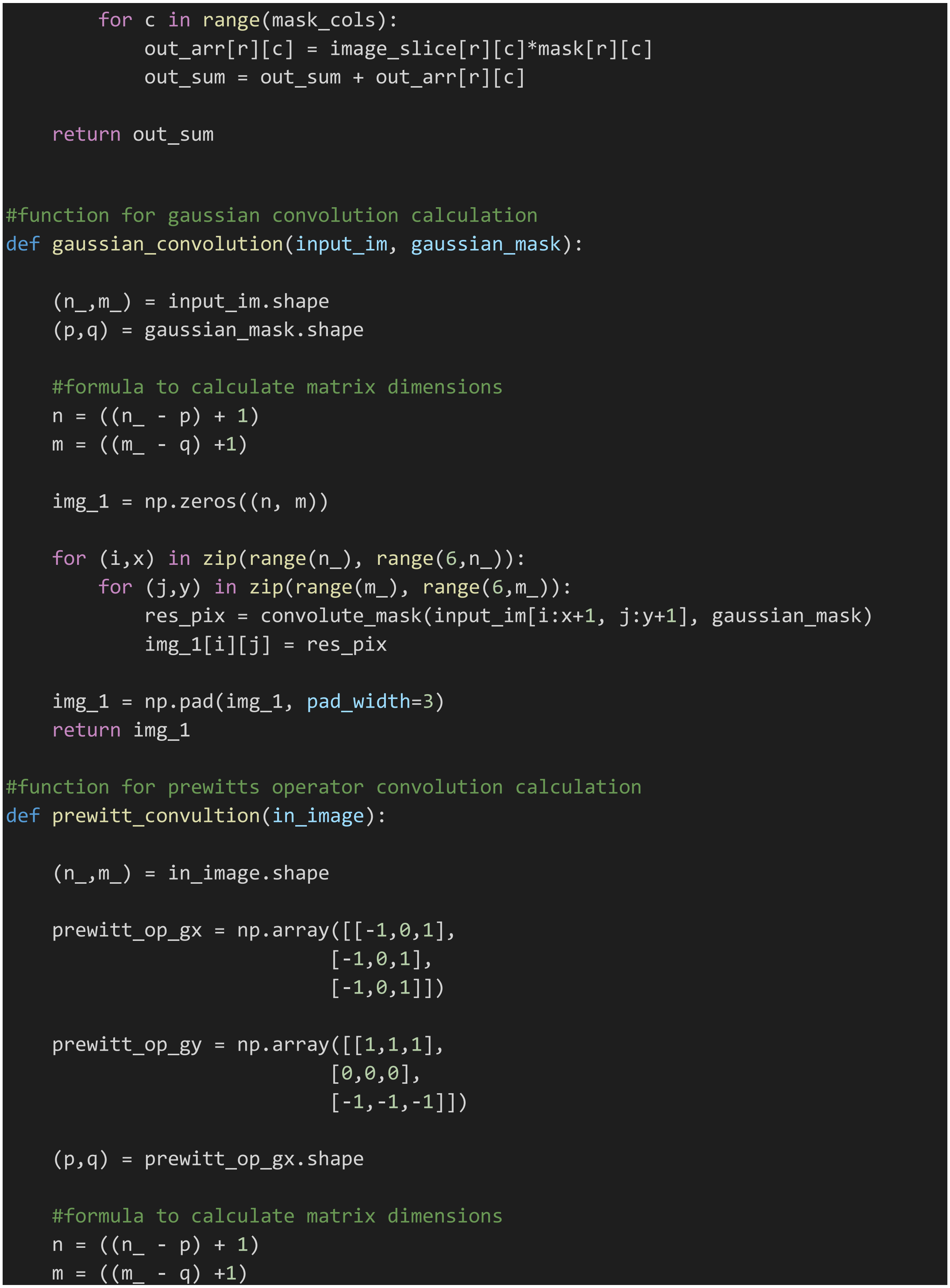
opencv-python, numpy, matplotlib

Source Code :

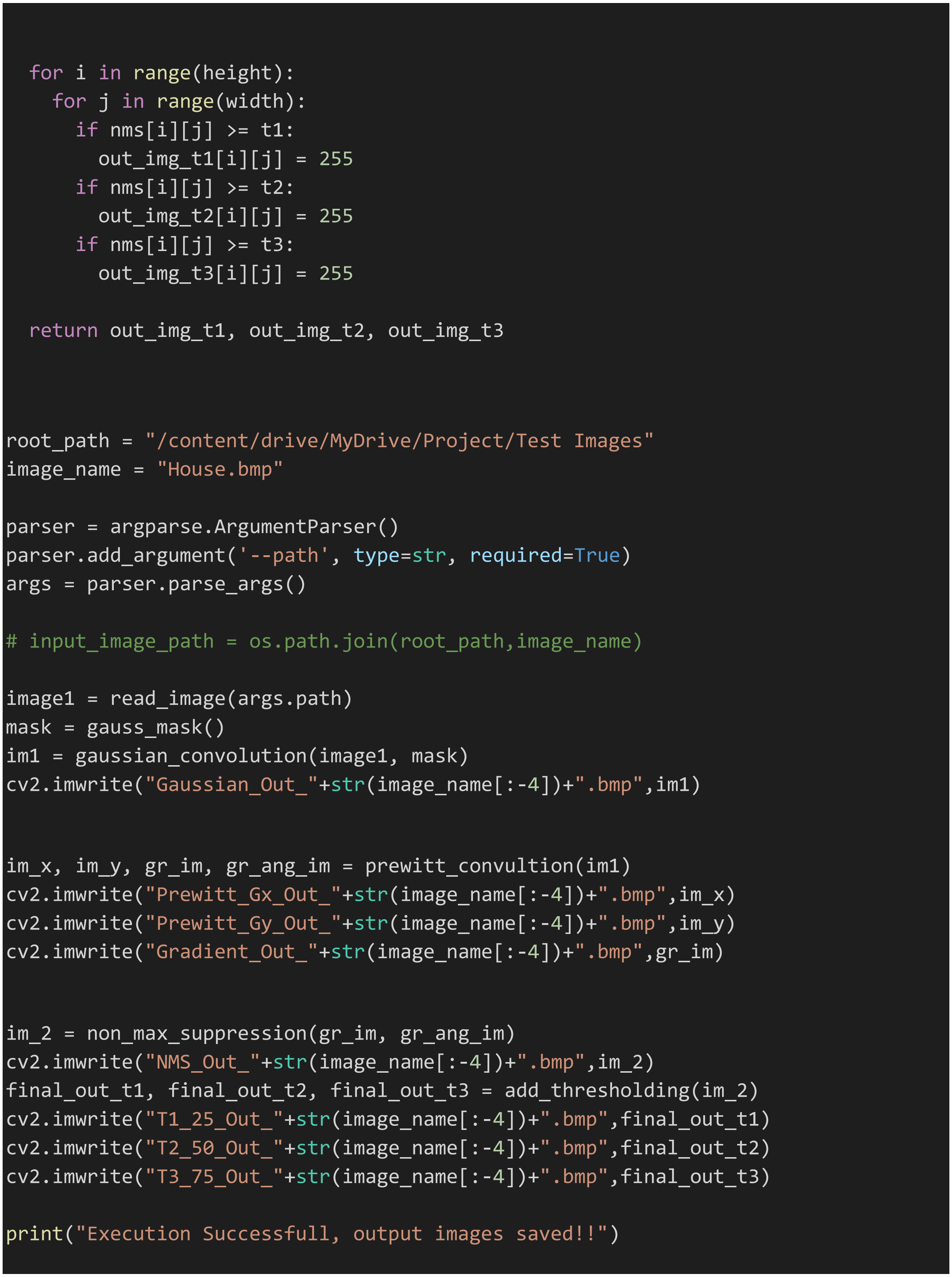
```
# Project 1 - Canny Edge Detection
# Student Name 1 - Harsh Sanjay Apte, NetID - ha2179
# Student Name 2 - Savani Manoj Gokhale, NetID - sg6428


from matplotlib import pyplot as plt
# from PIL import Image
import numpy as np
# from google.colab.patches import cv2_imshow
import cv2
import os
import argparse

# function to read image
```



```
# parameters ==> img_path = image path
# returns ==> image matrix

def read_image(img_path):
  image = cv2.imread(img_path)
  image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
  return image

#function to generate gaussian mask
def gauss_mask():
  gaussian_mask = np.array([[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]])
  return gaussian_mask


#function for convoluting gaussian mask over image
def convolute_mask(image_slice, mask):

    mask_rows, mask_cols = mask.shape
    out_arr = np.zeros((mask_rows, mask_cols))
    out_sum = 0

    for r in range(mask_rows):
        for c in range(mask_cols):
            out_arr[r][c] = image_slice[r][c]*mask[r][c]
            out_sum = out_sum + out_arr[r][c]

    #normalization of pixel values
    final_pix = out_sum/140
    return final_pix


#function for convoluting prewitt's operator over image
def convolute_mask_prewitt(image_slice, mask):

    mask_rows, mask_cols = mask.shape
    out_arr = np.zeros((mask_rows, mask_cols))
    out_sum = 0

    for r in range(mask_rows):
```



```
        for c in range(mask_cols):
            out_arr[r][c] = image_slice[r][c]*mask[r][c]
            out_sum = out_sum + out_arr[r][c]

    return out_sum


#function for gaussian convolution calculation
def gaussian_convolution(input_im, gaussian_mask):

    (n_,m_) = input_im.shape
    (p,q) = gaussian_mask.shape

    #formula to calculate matrix dimensions
    n = ((n_ - p) + 1)
    m = ((m_ - q) +1)

    img_1 = np.zeros((n, m))

    for (i,x) in zip(range(n_), range(6,n_)):
        for (j,y) in zip(range(m_), range(6,m_)):
            res_pix = convolute_mask(input_im[i:x+1, j:y+1], gaussian_mask)
            img_1[i][j] = res_pix

    img_1 = np.pad(img_1, pad_width=3)
    return img_1

#function for prewitts operator convolution calculation
def prewitt_convultion(in_image):

    (n_,m_) = in_image.shape

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

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

    (p,q) = prewitt_op_gx.shape

    #formula to calculate matrix dimensions
    n = ((n_ - p) + 1)
    m = ((m_ - q) +1)
    img_gx = np.zeros((n,m))
    img_gy = np.zeros((n,m))

    for (i,x) in zip(range(n_), range(2,n_)):
        for (j,y) in zip(range(m_), range(2,m_)):
            res_pix_x = convolute_mask_prewitt(in_image[i:x+1, j:y+1],
prewitt_op_gx)
            img_gx[i][j] = res_pix_x

            res_pix_y = convolute_mask_prewitt(in_image[i:x+1, j:y+1],
prewitt_op_gy)
            img_gy[i][j] = res_pix_y

    img_gx = np.pad(img_gx, pad_width=1)
    img_gy = np.pad(img_gy, pad_width=1)

    #gradient calculation
    gradient_img_out = np.add(np.absolute(img_gx), np.absolute(img_gy))

    #gradient angle calculation
    gradient_angle = np.arctan2(img_gy, img_gx)

    return img_gx, img_gy, gradient_img_out, gradient_angle

#function for non-max suppression
def non_max_suppression(gradient_img, gradient_angle):
    (n, m) = gradient_img.shape

    for i in range(1, n-1):
        for j in range(1, m-1):

            # 0 in angle comparison pie
            if gradient_angle[i][j] >= -22.5 and gradient_angle[i][j] < 22.5:
                if max(gradient_img[i][j], gradient_img[i][j-1],
gradient_img[i][j+1]) != gradient_img[i][j]:
                    gradient_img[i][j] = 0

            # 1 in angle comparison pie
            elif gradient_angle[i][j] >= 22.5 and gradient_angle[i][j] < 67.5:
                if max(gradient_img[i][j], gradient_img[i-1][j+1],
gradient_img[i+1][j-1]) != gradient_img[i][j]:
                    gradient_img[i][j] = 0

            # 2 in angle comparison pie
            elif gradient_angle[i][j] >= 67.5 and gradient_angle[i][j] < 90:
                if max(gradient_img[i][j], gradient_img[i-1][j],
gradient_img[i+1][j]) != gradient_img[i][j]:
                    gradient_img[i][j] = 0

            # 2 in angle comparison pie
            elif gradient_angle[i][j] >= -90 and gradient_angle[i][j] < -67.5:
                if max(gradient_img[i][j], gradient_img[i-1][j],
gradient_img[i+1][j]) != gradient_img[i][j]:
                    gradient_img[i][j] = 0

            # 3 in angle comparison pie
            elif gradient_angle[i][j] >= -67.5 and gradient_angle[i][j] < -22.5:
                if max(gradient_img[i][j], gradient_img[i-1][j-1],
gradient_img[i+1][j+1]) != gradient_img[i][j]:
                    gradient_img[i][j] = 0

    return gradient_img


#simple thresholding function
def add_thresholding(nms):
  nms_flatten = nms.flatten()
  flatten_len = len(nms_flatten)

  # logic to pick only non zero values in image
  non_zero_pix = []
  for i in range(flatten_len):
      pix_value = nms_flatten[i]
      if pix_value != 0:
        non_zero_pix.append(pix_value)

  non_zero_pix = sorted(non_zero_pix, reverse=True)

  #percentile calculation
  t1 = np.percentile(non_zero_pix, 25)

  t2 = np.percentile(non_zero_pix, 50)

  t3 = np.percentile(non_zero_pix, 75)

  height, width = nms.shape
  out_img_t1 = np.zeros((height,width))
  out_img_t2 = np.zeros((height,width))
  out_img_t3 = np.zeros((height,width))
```



```
  for i in range(height):
    for j in range(width):
      if nms[i][j] >= t1:
        out_img_t1[i][j] = 255
      if nms[i][j] >= t2:
        out_img_t2[i][j] = 255
      if nms[i][j] >= t3:
        out_img_t3[i][j] = 255

  return out_img_t1, out_img_t2, out_img_t3



root_path = "/content/drive/MyDrive/Project/Test Images"
image_name = "House.bmp"

parser = argparse.ArgumentParser()
parser.add_argument('--path', type=str, required=True)
args = parser.parse_args()

# input_image_path = os.path.join(root_path,image_name)

image1 = read_image(args.path)
mask = gauss_mask()
im1 = gaussian_convolution(image1, mask)
cv2.imwrite("Gaussian_Out_"+str(image_name[:-4])+".bmp",im1)


im_x, im_y, gr_im, gr_ang_im = prewitt_convultion(im1)
cv2.imwrite("Prewitt_Gx_Out_"+str(image_name[:-4])+".bmp",im_x)
cv2.imwrite("Prewitt_Gy_Out_"+str(image_name[:-4])+".bmp",im_y)
cv2.imwrite("Gradient_Out_"+str(image_name[:-4])+".bmp",gr_im)


im_2 = non_max_suppression(gr_im, gr_ang_im)
cv2.imwrite("NMS_Out_"+str(image_name[:-4])+".bmp",im_2)
final_out_t1, final_out_t2, final_out_t3 = add_thresholding(im_2)
cv2.imwrite("T1_25_Out_"+str(image_name[:-4])+".bmp",final_out_t1)
cv2.imwrite("T2_50_Out_"+str(image_name[:-4])+".bmp",final_out_t2)
cv2.imwrite("T3_75_Out_"+str(image_name[:-4])+".bmp",final_out_t3)

print("Execution Successfull, output images saved!!")
```

Output Images in Sequence of –

1) Gaussian Filtering

2) Gradient Magnitude

3) Non-Max Suppression

4) Prewitt Operator Gx Output

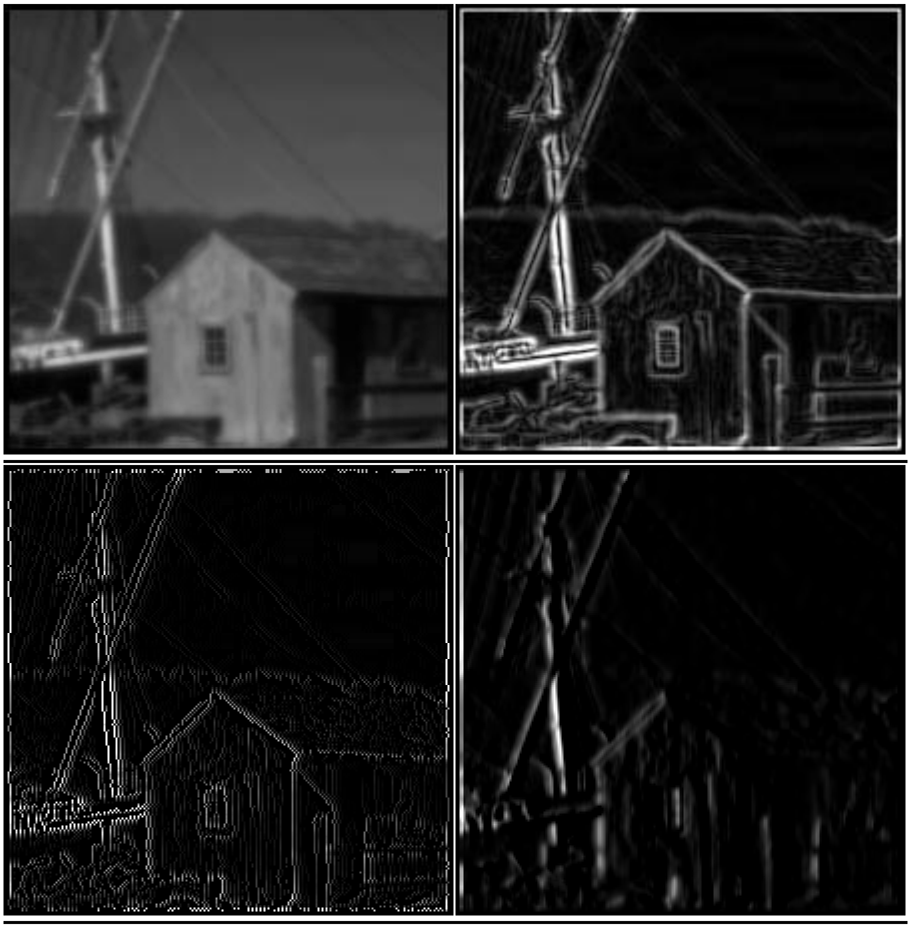
5) Prewitt Operator Gy Output

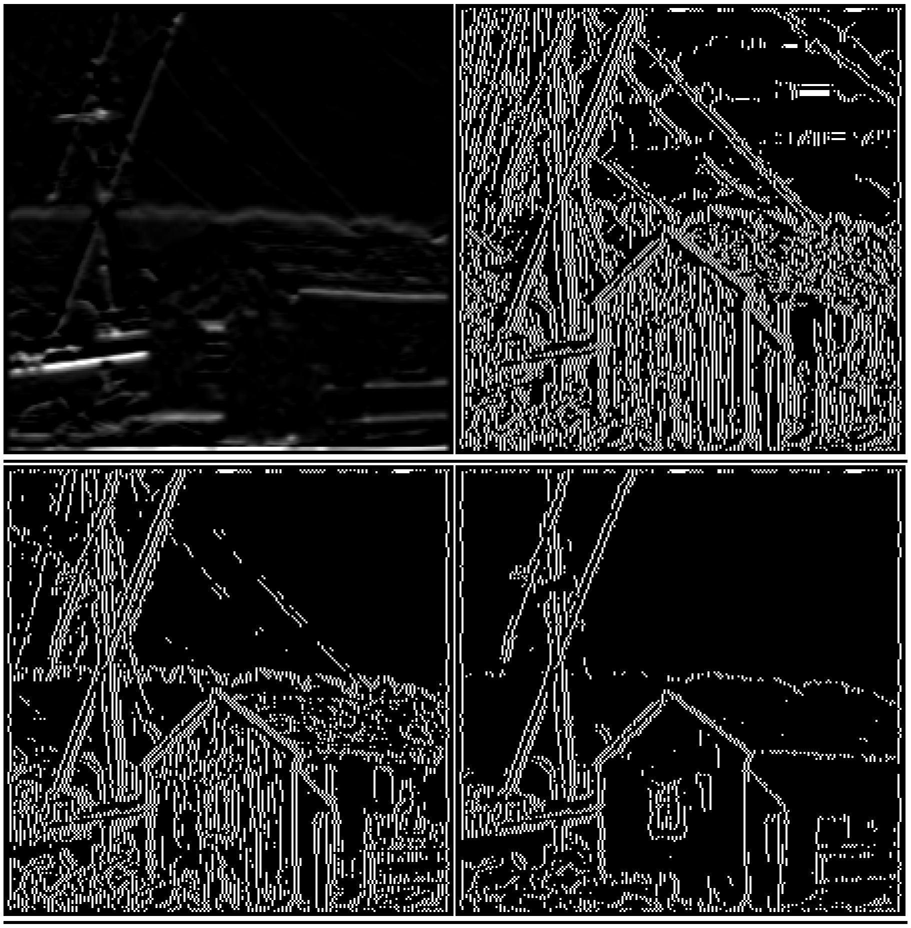
6) 25 Percentile Threshold

7) 50 Percentile Threshold

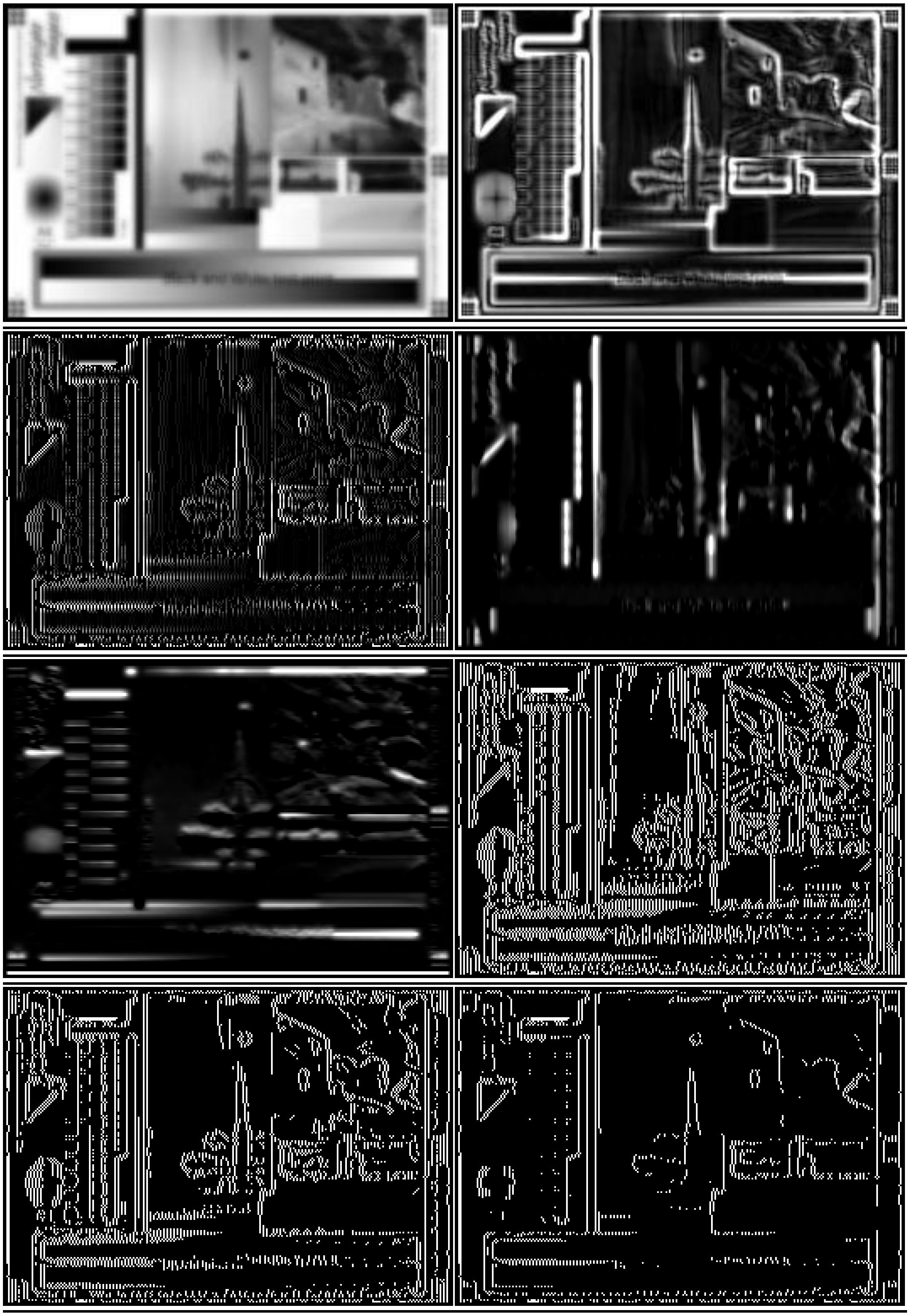
8) 75 Percentile Threshold

1) <u>House.bmp Output Images :</u>





2) <u>Test patterns.bmp Output Images:</u>


Black and White test print