Computer Vision Project 1
Canny Edge Detector
netid - haj272
Hardik Jivani

(i) File name of your source code.
canny_edge_detector.py

(ii) Instructions on how to run your program and instructions on how to compile your program if your program requires compilation.

Create python virtual environment by installing packages such as opencv-python, numpy, math and sys

Execute this script file using python command

python3 script_file.py {T1} {T2}

Example:
python3 canny_edge_detector.py 5 10

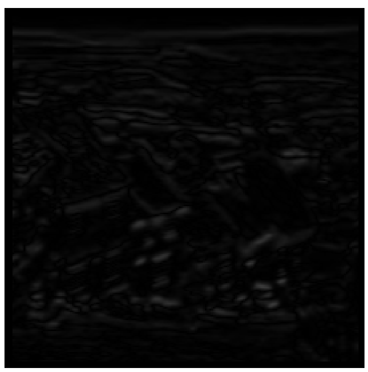
(iii) Output image results (1) to (5) for all test images.

1. Houses-225.bmp

a) Gaussian output

b) Horizontal Normalized Gradient

c) Vertical Normalized Gradient



d) Normalized Gradient Magnitude

e) Non Maxima Suppresion Output

f) Double Thresholding output with T1 = 5 and T2 = 10

2) Zebra-crossing-1.bmp

a) Gaussian Output

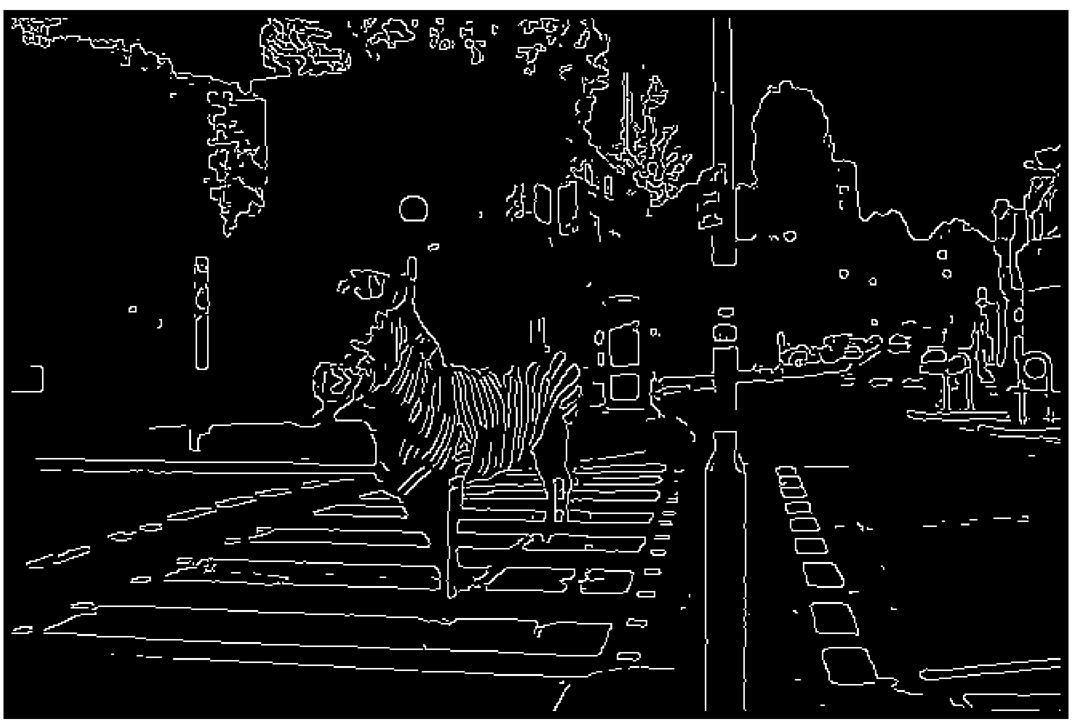
b) Normalized Horizontal Gradient Output

c) Normalized Vertical Gradient Output

d) Normalized Gradient Magnitude

e) Non Maxima Suppressed Output

f) Double Thresholding Output with **T1 = 8 and T2 = 16**



(4) Source code

```
'''
Source code for edge detection using Canny Edge Detector
'''
import numpy as np
import sys
import cv2
import math

def compute_convolution(img):
    '''
    Function to compute convolution of image with gaussian filter
    '''
    #define a new matrix of the size of image and initialize all pixels with zero
    convolution_result = np.zeros(shape=img.shape)

    #initializa gaussian mask of size 7x7 normalizing value by
    #dividing all the elements with 140, as sum of all the pixels in the mask i
    mask = (1.0/140.0)*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)])

    #perform convolution and store the result in "convolution_result"
    for row in range(img.shape[0]-6):
        for col in range(img[row].size-6):
            filter_sum = 0
            for i in range (0,7):
                for j in range (0,7):
                    filter_sum = filter_sum + img[row+i,col+j]*mask[i,j]
            convolution_result[row+3,col+3] = filter_sum
    return convolution_result

def compute_gradient_results(img):
    '''
    Function to compute normalized horizontal gradient, normalized vertical gradient,
    Normalized Gradient Magnitude and Gradient Angle
    '''
    #Sobel Filter Mask to calculate Horizontal(x) gradient
    sobel_x = (1/4)*np.array([(-1,0,1),
            (-2,0,2),
```

```
        (-1,0,1)])
#Sobel Filter Mask to calculate Vertical(y) gradient
sobel_y = (1/4)*np.array([(1,2,1),
        (0,0,0),
        (-1,-2,-1)])

#initialize matrices to store the value of horizontal and vertical gradient,
#normalized horizontal and vertical gradient, normalized gradient magnitude
#and gradient angle
gradientx = np.zeros(shape=img.shape)
gradientx_normalized = np.zeros(shape=img.shape)
gradienty = np.zeros(shape=img.shape)
gradienty_normalized = np.zeros(shape=img.shape)
gradient_magnitude = np.zeros(shape=img.shape)
gradient_angle = np.zeros(shape=img.shape)

#find the gradient values by perfoeming convolution
for row in range(4,img.shape[0]-5):
  for col in range(4,img[row].size-5):
    #calculate Value at current pixel (row,col)
    #after applying sobel operator
    sobel_gx = 0
    sobel_gy = 0
    for i in range (0,3):
      for j in range (0,3):
        sobel_gx = sobel_gx + img[row+i,col+j]*sobel_x[i,j]
        sobel_gy = sobel_gy + img[row+i,col+j]*sobel_y[i,j]
    gx = sobel_gx
    gradientx[row+1,col+1] = gx
    #calculate normalized horizontal gradient
    gradientx_normalized[row+1,col+1] = abs(gx)
    gy = sobel_gy
    gradienty[row+1,col+1] = gy
    #calculate normalized vertical gradient
    gradienty_normalized[row+1,col+1] = abs(gy)
    #normalize gradient magnitude by dividing by sqrt(2)
    gradient_magnitude[row+1,col+1]=((gx**2+gy**2)**(0.5))/(1.4142)
    #calculate gradient angle based on sobel horizontal gradient and vertical gradient
    angle = 0
    if(gx == 0):
      if( gy > 0):
        angle = 90
      else:
        angle = -90
```

```
            else:
                angle = math.degrees(math.atan(gy/gx))
            if (angle < 0):
                angle = angle + 360
            gradient_angle[row+1,col+1]  = angle
    return [gradientx_normalized, gradienty_normalized, gradient_magnitude, gradient_angle]

def compute_nonMaximaSuppressed_image(gradient, gradient_angle):
    '''
    function to perform non maxima suppression
    gradient = a matrix containing gradient values at each pixel of the image
    gradient_angle = a matrix containing gradient angle at each pixel of the image
    '''
    #initialize matrix with zeros to store the output of non maxima suppression
    nms = np.zeros(shape=gradient.shape)
    for row in range(5,gradient.shape[0]-5):
        for col in range(5,gradient[row].size-5):
            angle = gradient_angle[row,col]
            # gradient at current pixel
            curr = gradient[row,col]
            val = 0
            #sector zero
            if( 0 <= angle <= 22.5 or  157.5 < angle <= 202.5 or 337.5 < angle <= 360):
                val = curr if (curr > gradient[row,col+1] and curr > gradient[row,col-1]) else 0
            #sector one
            elif ( 22.5 < angle <= 67.5 or  202.5 < angle <= 247.5):
                val = curr if (curr > gradient[row+1,col-1] and curr > gradient[row-1,col+1]) else 0
            #sector two
            elif ( 67.5 < angle <= 112.5 or  247.5 < angle <= 292.5):
                val = curr if (curr > gradient[row+1,col] and curr > gradient[row-1,col]) else 0
            #sector three
            elif ( 112.5 < angle <= 157.5 or  292.5 < angle <= 337.5):
                val = curr if (curr > gradient[row+1,col+1] and curr > gradient[row-1,col-1]) else 0
            nms[row,col] = val
    return nms

def compute_double_thresholding(nms, gradient_angle, T1, T2):
    '''
    Function to compute double thresholding on the normalizaed non maxima suppressed image
    '''
    # initialize array with zeros to store threshold output calculated from non maximum
suppression output
    dt_img = np.zeros(shape=nms.shape)
    for row in range(5,nms.shape[0]-5):
```

```
        for col in range(5,nms[row].size-5):
            # check if the current pixel value is less than T1
            # then assign 0 in the output
            if nms[row,col] < T1:
                dt_img[row,col] = 0
            # check if the current pixel value is greater than T2
            # then assign 255 in the output
            elif nms[row,col] > T2:
                dt_img[row,col] = 255
            # check if the current pixel value is in between T1 and T2
            # then check 8 neighbor of the current pixel
            else:
                curr = gradient_angle[row, col]
                done = False
                for i in range(row-1,row+2):
                    for j in range(col-1,col+2):
                        if i == row and j == col:
                            continue
                        if nms[i,j]>T2 and abs(gradient_angle[i,j]-curr)<=45:
                            dt_img[row,col] = 255
                            done =True
                            break
                    if done:
                        break
                if not done:
                    dt_img[row,col] = 0
    return dt_img

def save_output(file, con, gradient_x, gradient_y, gradient, nms, threshold):
    filename = file.split('.')[0]
    cv2.imwrite(filename+'_gaussian.bmp', con)
    cv2.imwrite(filename+"_gradientX.bmp",gradient_x)
    cv2.imwrite(filename+"_gradientY.bmp",gradient_y)
    cv2.imwrite(filename+"_gradient.bmp",gradient)
    cv2.imwrite(filename+"_nms.bmp",nms)
    cv2.imwrite(filename+"_thresholding.bmp",threshold)

def main():
    file = sys.argv[1]
    T1 = int(sys.argv[2])
    T2 = int(sys.argv[3])
    image = cv2.imread(file,0)
    convolution_output = compute_convolution(image)
    gradients = compute_gradient_results(convolution_output)
```

```
    gradient_x = gradients[0]
    gradient_y = gradients[1]
    gradient_magnitude = gradients[2]
    gradient_angles = gradients[3]
    non_maximum_suppressed_output =
compute_nonMaximaSuppressed_image(gradient_magnitude, gradient_angles)
    double_threshold_output =
compute_double_thresholding(non_maximum_suppressed_output, gradient_angles, T1, T2)
    save_output(file, convolution_output, gradient_x, gradient_y, gradient_magnitude,
                non_maximum_suppressed_output, double_threshold_output)

if __name__ == '__main__':
    main()
```