(i)Text file containing source code – sourcecode.txt

(ii) Instructions on how to run the program –

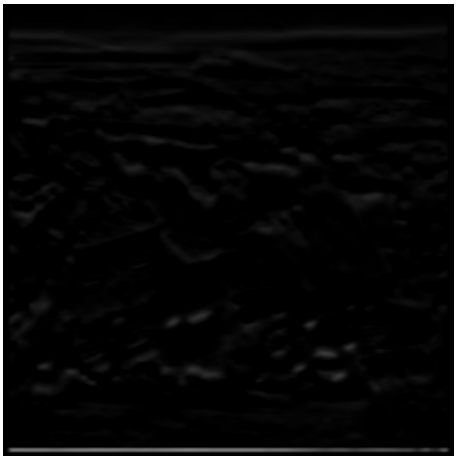
- Copy the code in any python compiler
- In main function parameters input the base directory where images are stored in the dir_name parameter
- In 6th line in main function where image is read input the name of the input image
- Set threshold t1,t2
- Run the code

(iii)

1)

2)hx

hy



3)

4)

5)

1)

2)hx

Hy

3)

4)

5)

v)Source Code-

```
import numpy as np
from scipy.ndimage.filters import convolve
import cv2


def gaussianfiltering(matrix, mask):
    (iH, iW) = matrix.shape[:2] #find the width and height of image
    output = np.zeros((iH, iW))
    for y in np.arange(3, iH - 3):
        for x in np.arange(3, iW - 3):
            #perform convolution on the 7*7 submatrices of the image matrix using the
gausian mask
            output[y, x] = ((matrix[y - 3:y + 3 + 1, x - 3:x + 3 + 1] * mask).sum())
// 140
    return output

#this is a function used to perform convolution on "matrix" using the convolution
mask "mask"
def convolve(matrix, mask):
    (iH, iW) = matrix.shape[:2]
    output = np.zeros((iH, iW))
    for y in np.arange(1, iH - 1):
        for x in np.arange(1, iW - 1):
            output[y , x] = ((matrix[y - 1:y + 2, x - 1:x +2] * mask).sum())
    return output


def gradientoperation(hx, hy):
    row, column = hx.shape
    #create a new array to store output magnitude array
    magnitudearray = np.hypot(hx, hy) #magnitude=sqrt(hx^2+hy^2)
    gradientangle = np.zeros((len(hx), len(hx[0])))
    for i in range(1, row-1):
        for j in range(1, column-1):
            if(hx[i][j]==0):
                gradientangle[i][j]=0 #gradient will be undefined if hx[i][j]==0 so
replace undefined to 0
            else:
                #gradient=tan^-1(Hy/Hx)
                gradientangle[i][j] = np.degrees(np.arctan(hy[i][j]/hx[i][j]))+180

    return (magnitudearray, gradientangle)

#this function firstly classifies each pixel to a sector(0,1,2,3) based on its the
gradient angle value
#later on depending on sector campare that pixels magnitude with its neighbours along
the sector line
```

```
# if greater than both then keep it as it is otherwise reduce to 0
def nonmaximasuppression(M, gradangle):
    sector = np.zeros((len(M), len(M[0])))
    N = np.zeros((len(M), len(M[0])))
    row, column = M.shape
    for i in range(1, row-1):
        for j in range(1, column-1):
            if ((0 <=gradangle.item(i, j) < 22.5) or (157.5<=gradangle.item(i, j) <
202.5) or(337.5<=gradangle.item(i, j)<=360)):
                sector[i, j] = 0
                if (M.item(i, j) > max(M.item(i, j + 1),M.item(i, j - 1))):
                    N[i][j] = M.item(i, j)
                else:
                    N[i][j] = 0
            elif ((22.5 <= gradangle.item(i, j) < 67.5) or (202.5 <=
gradangle.item(i, j) < 247.5)):
                sector[i, j] = 1
                if (M.item(i, j) > max(M.item(i - 1, j + 1),M.item(i + 1, j - 1))):
                    N[i][j] = M.item(i, j)
                else:
                    N[i][j] = 0
            elif ((67.5<= gradangle.item(i, j) < 112.5) or (247.5<= gradangle.item(i,
j) <= 292.5)):
                sector[i][j] = 2
                if (M.item(i, j) > max(M.item(i - 1, j),M.item(i + 1, j))):
                    N[i][j] = M.item(i, j)
                else:
                    N[i][j] = 0
            elif ((112.5 <= gradangle.item(i, j) < 157.5) or (292.5 <=
gradangle.item(i, j) < 337.5)):
                sector[i][j] = 3
                if (M.item(i, j) > max(M.item(i - 1, j - 1),M.item(i + 1, j + 1))):
                    N[i][j] = M.item(i, j)
                else:
                    N[i][j] = 0
    return N,gradangle

#here we compare individual pixel value with two thresholds t1&t2
#based on conditions as show below finalise the magnitude of the pixels and form the
edge map
def thresholding(magnitude, t1, t2,gradangle):
    row,column=magnitude.shape
    edgemap = np.zeros((row,column))
    for i in range(0, row - 1):
        for j in range(0, column - 1):
            if(magnitude[i,j]<t1):
                edgemap[i][j]=0
            elif(magnitude[i,j]>t2):
                edgemap[i][j]=255
            elif((magnitude[i,j]>=t1) & (magnitude[i,j]<=t2)):
                if (((magnitude[i + 1, j - 1] > t2) & (abs(gradangle[i + 1, j - 1]-
gradangle[i, j])<= 45)) or
                    ((magnitude[i + 1, j] > t2) & (abs(gradangle[i + 1, j]-
gradangle[i, j])<=45)) or
                    ((magnitude[i + 1, j + 1] > t2) & (abs(gradangle[i + 1, j + 1]-
```

```
gradangle[i, j])<=45))or
                    ((magnitude[i, j - 1] > t2) & (abs(gradangle[i, j - 1]-
gradangle[i, j])<=45)) or
                    ((magnitude[i, j + 1] > t2) & (abs(gradangle[i, j + 1]-
gradangle[i, j])<=45))or
                    ((magnitude[i - 1, j - 1] > t2) & (abs(gradangle[i - 1, j - 1]-
gradangle[i, j])<=45)) or
                    ((magnitude[i - 1, j] > t2) & (abs(gradangle[i - 1, j]-
gradangle[i, j])<=45)) or
                    ((magnitude[i - 1, j + 1] > t2) & (abs(gradangle[i - 1, j + 1]-
gradangle[i, j])<45))
                    ):
                    edgemap[i, j] = 255
                else:
                    edgemap[i, j] = 0
    return edgemap

def main(dir_name='faces_imgs'):
    #define gaussian mask,gx,gy,t1,t2 values to use later
    gaussianmask = 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]))
    gx = np.array(([-1, 0, 1], [-2, 0, 2], [-1, 0, 1]))
    gy = np.array(([1, 2, 1], [0, 0, 0], [-1, -2, -1]))
    t2 = 10
    t1 = 5

    #read image using cv2 library function
    greyimage = cv2.imread(dir_name + '/2.bmp',cv2.IMREAD_GRAYSCALE) #read image

    #function calls to all the above mentioned functions
    filteringoutput = gaussianfiltering(greyimage, gaussianmask) #pass the mask and
image as parameters
    cv2.imwrite(dir_name + '/(1)Gaussian.png', filteringoutput)

    hx = convolve(filteringoutput, gx)
    hy = convolve(filteringoutput, gy)

    #normalise hx & hy
    ohx = hx // 4
    ohy=hy//4
    #cv2.imwrite(dir_name + '/Hx.png',hx)
    #cv2.imwrite(dir_name + '/Hy.png',hy)
    cv2.imwrite(dir_name + '/(2)normalisedHx.png', ohx)
    cv2.imwrite(dir_name + '/(2)normalizedHy.png', ohy)
    #calculate magnitude array and gradient array
    magnitudearray, gradientangle = gradientoperation(ohx,ohy)
    normalisedgradientmagnitude= magnitudearray // np.sqrt(2)
    #cv2.imwrite(dir_name + '/magarray.png', magnitudearray)
    cv2.imwrite(dir_name + '/(3)normalisedgradientmagnitude.png',
normalisedgradientmagnitude)
    #perform nonmaximasupression
    nonmaximasupressionop,gradangle =
nonmaximasuppression(normalisedgradientmagnitude, gradientangle)
```

```
    cv2.imwrite(dir_name + '/(4)nonmaxima.png', nonmaximasupressionop)
    #double thresholding
    edgemap= thresholding(nonmaximasupressionop, t1, t2,gradangle)
    cv2.imwrite(dir_name + '/(5)edgemap.png', edgemap)

main()
```