ARJAN SINGH NARULA

N15292730

asn419@nyu.edu

CV Project 1

1.Source Code: ARJAN_Canny_Edge_Detector.txt

Python File-ARJAN_Canny_Edge_Detector.py

2.Instructions

Source Code: To Compile and Run the program follow these steps.

- Copy the source code in python compiling platform like Anaconda(Jupyter)or Pycharm.
- Make sure system has Python Interpreter installed**.**
- Before compiling, make sure that the libraries 'numpy', 'Imageio', 'PIL' and 'matplotlib' are linked to the project
- After running the code, it will ask you to input the name of image with extension. Image should be on the same folder as the code.
- The code will run and save the output images in the same folder ARJAN_Edge_Detector with name of images.

3.Output Image Results

A. Lena256

- **Normal Image**

- **Gaussian Smoothed Image**

Fig(1a): Gaussian Smoothed Image

- **Gradient X**

Fig(2a):Gradient X Image

- **Gradient Y**

Fig(3a): Gradient Y Image

- **Magnitude Image**

Fig(4a):Magnitude Image

- **Non Maxima Suppression**

Fig(5a):Non Maxima Suppression

- **Ptile=10 Percent**

Fig(6a1):Image for 10 Percent

Edges Detected: 985
Threshold: 30

- **Ptile=30 Percent**

Fig(6a2): Image for 30 Percent

Edges Detected: 2657
Threshold: 15

- Ptile=50 Percent

Fig(6a3): Ptile for 50 Percent

Edges Detected: 4612
Threshold: 6

B. zebra-crossing-1

- **Gaussian Smooth Image**

Fig(1b):Gaussian Smooth Image

- **Gradient X**

Fig(2b):Gradient X

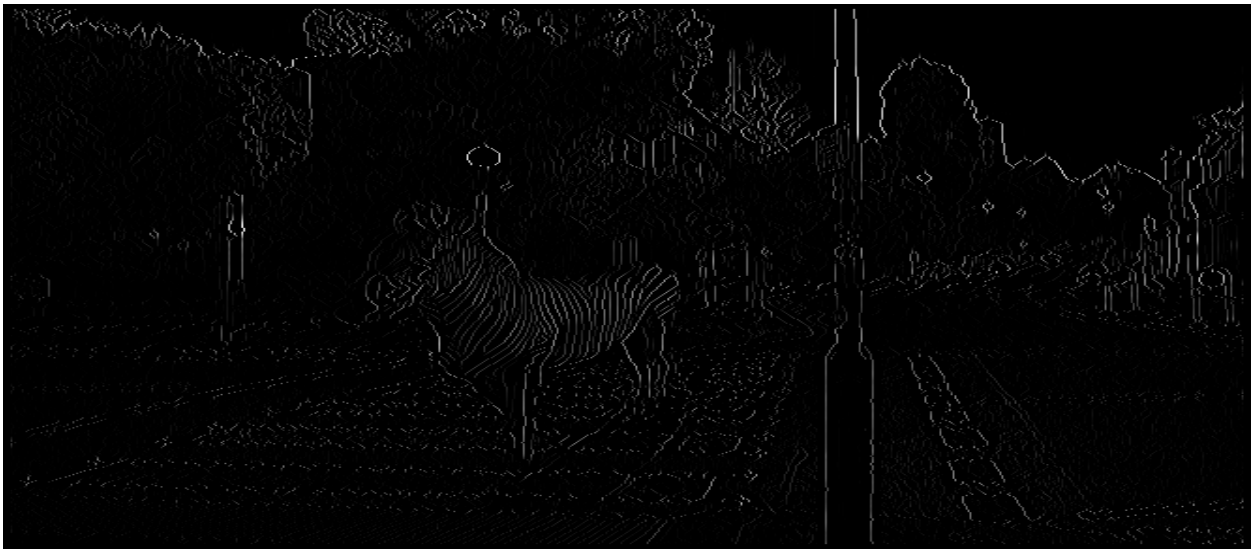
- **Gradient Y**

Fig(3b):Gradient Y

- **Magnitude Image**

Fig(4b):Magnitude Image

- **Non Maxima Suppression**



Fig(5b):Non Maxima Suppression

- **Ptile=10Percent**

Fig(6b1): Ptile for 10 Percent

Edges Detected: 3789
Threshold: 29

- **Ptile=30 Percent**

Fig(6b2): Ptile for 30 Percent

Edges Detected: 11368
Threshold: 12

- **Ptile=50 Percent**

Fig(6b3): Ptile for 50 Percent

Edges Detected: 18819
Threshold: 5

//Source Code

```
from PIL import Image
import numpy as np
from math import sqrt
from math import degrees
import imageio as io
def main():
#   Opening image
    Imagename=input("Give name of image with extension : ")
    img=np.array(Image.open(Imagename))
#   Rows and Coloumn of image
    x,y=img.shape
#   Gaussian convolution mask array

gaussian=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]])
#   After gaussian smoothed array till now its value are zero as we have not call our gaussian smooth
function
    gaussiansmootharray= np.zeros((x,y),dtype=np.float)
#   Row and coloumn of gaussian mask
    rowgaussian,colgaussian=gaussian.shape
#   Finding the medium point of gaussian mask for wor and coloumn
    rowmedian=rowgaussian//2
    colmedian=colgaussian//2
#   Called gaussian smoothing function for gaussian smoothing wih image array,gaussian array and
gaussian smooth array which is empty
    gaussiansmoothing(img,gaussian,gaussiansmootharray,Imagename)
#   Array with same size of image but with values 0 for gradient x and gradient y
    gx=np.zeros((x,y),dtype=np.float)
    gy=np.zeros((x,y),dtype=np.float)
```

```
#   Array with same size of image for gradient x and gradient y with values 0 but it is used to contains negative value too for angle array.
    gx1=np.zeros((x,y),dtype=np.float)
    gy1=np.zeros((x,y),dtype=np.float)
#   Call gradient operator function for calculating gradient x an gradient y
    gradientoperator(gaussiansmootharray,gx,gy,gx1,gy1,Imagename )
#   Magnitude array initialize for storing magnitudes of image after gradient operation
    magnitduearray=np.zeros((x,y),dtype=np.float)
#   Angle array for storing angles after gradient operation
    anglearray=np.zeros((x,y),dtype=np.float)
#   Call gradient magnitude function for calculating the magnitudes and angle of intensities.
    gradientmagnitude(gx1,gy1,magnitduearray,Imagename)
#   Initialize array for storing results after non maxima suppression
    after_nonmaximasppression=np.zeros((x,y),dtype=np.float)
#   Call non maxima suppression function to calulate non maxima.
    non_maxima_suppression(magnitduearray,anglearray,after_nonmaximasppression,Imagename)
#   Taking the result of non maxima suppression and applying it to ptile function for calulation of ptile, threshold and detecting the number of edges in a image
    ptile1=[0.1,0.3,0.5]
    for i in range(len(ptile1)):
        print("Ptile for",ptile1[i]*100,"percent")
        ptile(after_nonmaximasppression,ptile1[i],Imagename)
```

```
def multiply(imgarr,gaussianarr,startrow,startcol,gaussianstartrow,gaussianstartcol):
    sum=0
    i=startrow
    j=startcol
    e=gaussianstartrow
```

```
    f=gaussianstartcol


    for a in range(e):
        j=startcol


        for b in range(f):
            c=imgarr[i][j]*gaussianarr[a][b]
            sum=sum+c
            j=j+1
        i=i+1
    return sum
def gaussiansmoothing(img,gaussian,gaussiansmootharray,Imagename):
#   Row and Column of Image
    x,y=img.shape
#   Row and Column of Gaussian Mask
    rowgaussian,colgaussian=gaussian.shape
#   Medium for Row and Coloumn of Gaussian Mask
    rowmedian=rowgaussian//2
    colmedian=colgaussian//2
#   Works will i and j are in range of rows and coloumn for image
    for i in range(x):
        for j in range(y):
#  checking whether the above or below pixels or above pixel are in range of Gaussian Mask so that my gaussian filter doesnot go outside the image.
            d=i-rowmedian
            e=i+rowmedian
            if((d)in range(x)):
                if((e)in range(x)):
#  checking whether the left or right side pixel are in range of Gaussian Mask so that my gaussian filter doesnot go outside the image.
                    f=j-colmedian
```

```
            h=j+colmedian
            if((f)in range(y)):
                if((h)in range(y)):
# if they are not outside range, we will send the image array with gaussian mask and starting range of both array for convolution and store the result at this point.
                    gaussiansmootharray[i][j]=multiply(img,gaussian,i-rowmedian,j-colmedian,rowgaussian,colgaussian)
# Normalizing the convolution result by dividing by 40.
                    gaussiansmootharray[i][j]=gaussiansmootharray[i][j]/140
# Display image after Gaussian Smoothing
    import imageio as i
    i.imwrite("Gaussiam Smoooth_ %s.bmp"%Imagename,gaussiansmootharray)
    print("Gaussian Smooth Image")
    %matplotlib inline
    from matplotlib import pyplot as plt
    plt.imshow(gaussiansmootharray,cmap="gray")
    plt.show()


def gradientoperator(gaussiansmootharray1,gx,gy,gx1,gy1,Imagename):

gaussian=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]])
    rowgaussian,colgaussian=gaussian.shape
    rowmedian=rowgaussian//2
    colmedian=colgaussian//2
# Creating Prewitt Operator Sx and Sy for calculating Gradient X and Gradient Y
    perwittoperatorsx=np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
# Size of Sx
    perwittoperatorsxx,perwittoperatorsxy=perwittoperatorsx.shape
    perwittoperatorsy=np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
# Size of Sy
```

```
    perwittoperatorsxx1,perwittoperatorsxy1=perwittoperatorsy.shape
    x1,y1=gaussiansmootharray1.shape
# Medium of row and coloumn for Prewitt Operator
    rowmediangradient=perwittoperatorsx/2
    colmediangradient=perwittoperatorsx/2
#Defining the range till when the prewitt operator convolution works.
    looprowrange=x1-rowgaussian+rowmedian
    loopcolrange=y1-colgaussian+colmedian
# Starting point for storing the result after prewitt operation
    a=rowgaussian//2+1
    b=colgaussian//2+1
    while(a<looprowrange):
        b=colgaussian//2+1
        while(b<loopcolrange):
# If my values are in range, it will call the multiply funciton for convolution between the prewitt Sx and Gaussian Smooth Array.
            gx[a][b]=multiply(gaussiansmootharray1,perwittoperatorsx,a-1,b-1,perwittoperatorsxx,perwittoperatorsxy)
# Storing the result in Gx1 for negative values for angle array.
            gx1[a][b]=gx[a][b]
#  Using absolute values and normalize the result.
            gx[a][b]=abs(gx[a][b])
            gx1[a][b]=gx1[a][b]/3
            gx[a][b]=gx[a][b]/3
# # If my values are in range, it will call the multiply funciton for convolution between the prewitt Sy and Gaussian Smooth Array.
            gy[a][b]=multiply(gaussiansmootharray1,perwittoperatorsy,a-1,b-1,perwittoperatorsxx1,perwittoperatorsxy1)
#Storing the Result in Gy1 for negative value for angle array.
            gy1[a][b]=gy[a][b]
# Using absolute values and normalizing the result.
```

```
            gy[a][b]=abs(gy[a][b])
            gy[a][b]=gy[a][b]/3
            gy1[a][b]=gy1[a][b]/3
            b=b+1
        a=a+1
#Output Image after Gradient Operation for Gx and Gy.
    print("Gradient X")
    import imageio as i
    i.imwrite("GradientX_ %s.bmp"%Imagename,gx)
    i.imwrite("GradientY_ %s.bmp"%Imagename,gy)
    %matplotlib inline
    from matplotlib import pyplot as plt
    plt.imshow(gx,cmap="gray")
    plt.show()
    print("Gradient Y")
    plt.imshow(gy,cmap="gray")
    plt.show()

def gradientmagnitude(gradientx, gradienty,magnitduearray,Imagename):
#Rows and Column for Gradient X.
    x,y=gradientx.shape
#Checking that variable are in range of array.
    for a in range(x):
        for b in range(y):
#Calculating the magnitude by taking Gx,Gy at this point and normalizing the result.
            magnitduearray[a][b]=((gradientx[a][b]*gradientx[a][b])+(gradienty[a][b]*gradienty[a][b]))
            magnitduearray[a][b]=sqrt(magnitduearray[a][b])
            magnitduearray[a][b]=magnitduearray[a][b]/1.4142
            c=gradienty[a][b]/gradientx[a][b]
# calculating the angle at each point and converting the result into degrees.
```

```
    anglearray=np.arctan2(gradienty,gradientx)
    x,y=gradientx.shape
    for e in range(x):
        for l in range(y):
            anglearray[e][l]=degrees(anglearray[e][l])

# Displaying the image after Magnitude and Angle Array calculation.
    print("Magnitude")
    import imageio as i
    i.imwrite("Magnitude_ %s.bmp"%Imagename,magnitduearray)
    %matplotlib inline
    from matplotlib import pyplot as plt
    plt.imshow(magnitduearray,cmap="gray")
    plt.show()

def non_maxima_suppression(magnitudearray,anglearray1,after_nonmaximasppression,Imagename):
# Storing the rows and column of Magnitude Array.
    x,y=magnitudearray.shape
# Start point for computation.
    i=1
    j=0
# Defining the regions for angle Like 0,1 or 2.
    a=22.5
    b=a+45
    c=b+45
    d=c+45
    e=d+45
    f=e+45
    g=f+45
    h=g+45
```

```
    o=h+22.5


    while(i<(x-1)):
        j=1
        while(j<(y-1)):
# Checking for each point which region my angle belongs to and on basis of that extracting values of magnitude and verify it is greater or not.If not we will make this point to 0.
            if(anglearray1[i][j]<a):
                if((magnitudearray[i][j]<magnitudearray[i][j-1]) or (magnitudearray[i][j]<magnitudearray[i][j+1])):
                    after_nonmaximasppression[i][j]=0
                    j=j+1
                else:
                    after_nonmaximasppression[i][j]=magnitudearray[i][j]
                    j=j+1
            elif(anglearray1[i][j]<b):
                if((magnitudearray[i][j]<magnitudearray[i-1][j+1]) or (magnitudearray[i][j]<magnitudearray[i+1][j-1])):
                    after_nonmaximasppression[i][j]=0
                    j=j+1
                else:
                    after_nonmaximasppression[i][j]=magnitudearray[i][j]
                    j=j+1
            elif(anglearray[i][j]<c):
                if((magnitudearray[i][j]<magnitudearray[i-1][j]) or (magnitudearray[i][j]<magnitudearray[i+1][j])):
                    after_nonmaximasppression[i][j]=0
                    j=j+1
                else:
                    after_nonmaximasppression[i][j]=magnitudearray[i][j]
                    j=j+1
```

```
        elif(anglearray1[i][j]<d):
            if((magnitudearray[i][j]<magnitudearray[i-1][j-
1])or(magnitudearray[i][j]<magnitudearray[i+1][j+1])):
                after_nonmaximasppression[i][j]=0
                j=j+1
            else:
                after_nonmaximasppression[i][j]=magnitudearray[i][j]
                j=j+1
        elif(anglearray1[i][j]<e):
            if((magnitudearray[i][j]<magnitudearray[i][j-1]) or
(magnitudearray[i][j]<magnitudearray[i][j+1])):
                after_nonmaximasppression[i][j]=0
                j=j+1
            else:
                after_nonmaximasppression[i][j]=magnitudearray[i][j]
                j=j+1
        elif(anglearray1[i][j]<f):
            if((magnitudearray[i][j]<magnitudearray[i-
1][j+1])or(magnitudearray[i][j]<magnitudearray[i+1][j-1])):
                after_nonmaximasppression[i][j]=0
                j=j+1
            else:
                after_nonmaximasppression[i][j]=magnitudearray[i][j]
                j=j+1
        elif(anglearray1[i][j]<g):
            if((magnitudearray[i][j]<magnitudearray[i-
1][j])or(magnitudearray[i][j]<magnitudearray[i+1][j])):
                after_nonmaximasppression[i][j]=0
                j=j+1
            else:
                after_nonmaximasppression[i][j]=magnitudearray[i][j]
```

```
                j=j+1
            elif(anglearray1[i][j]<h):
                if((magnitudearray[i][j]<magnitudearray[i-1][j-
1])or(magnitudearray[i][j]<magnitudearray[i+1][j+1])):
                    after_nonmaximasppression[i][j]=0
                    j=j+1
                else:
                    after_nonmaximasppression[i][j]=magnitudearray[i][j]
                    j=j+1
            elif(anglearray1[i][j]<o):
                if((magnitudearray[i][j]<magnitudearray[i][j-
1])or(magnitudearray[i][j]<magnitudearray[i][j+1])):
                    after_nonmaximasppression[i][j]=0
                    j=j+1
                else:
                    after_nonmaximasppression[i][j]=magnitudearray[i][j]
                    j=j+1
            else:
                print("error")
        i=i+1
#   Displaying the Image after Non Maxima Suppression.
    print("After Non Maxima Suppression")
    import imageio as i
    i.imwrite("Nomaxima_Suppression %s.bmp"%Imagename,after_nonmaximasppression)
    %matplotlib inline
    from matplotlib import pyplot as plt
    plt.imshow(after_nonmaximasppression,cmap="gray")
    plt.show()

def ptile(finaloutputarray,ptile,Imagename):
#  Rows and Column for final output.
```

```
    name1=ptile*100
    print(name1)
    x,y=finaloutputarray.shape
# Intializing my total number of pixels is zero.
    total_numberof_pixels=0
#Creating Histogram array from 0 to 255 for storing the count of pixels at these point of array.
    histogramarray=np.zeros(x,dtype=np.int)
# Calculating total number of pixels whose value is greater than 0 and incrementing the count of variable for intensity at subsequent point.
    for i in range(x):
        for j in range(y):
            if(int(finaloutputarray[i][j])>0):
                histogramarray[int(finaloutputarray[i][j])]=histogramarray[int(finaloutputarray[i][j])]+1

                total_numberof_pixels=total_numberof_pixels+1
# Calculating the Ptle for finding the threshold for an Image.
#     ptile1=int(0.5*total_numberof_pixels)
#     name1=ptile
    ptile=ptile*total_numberof_pixels
    foregrounpixel=255
# Start traversing from 255 to 0 till my ptile value becomes 0 as we need foreground. We will extract the total pixel at this intensity(point) and subtract from ptile till we reach 0.

    while(ptile>0):
        ptile=ptile-histogramarray[foregrounpixel]
        foregrounpixel=foregrounpixel-1

# As Ptile becomes 0 we will say from this point value and above will be my foreground.
        threshold=foregrounpixel+1

# Calculating Edges and finalizing the image on the basis of the threshold for the detection of Edges.
```

```
    edges=0
    i=0
    j=0
    for i in range(x):
        for j in range(y):
#If my value is less than threshold we will make this point 0 otherwise we will change the point intensity to 255 for the detection of edges.
            if(finaloutputarray[i][j]>threshold):
                finaloutputarray[i][j]=255
                edges=edges+1
#Result for Edges and Threshold.
    print("Edges Detected",edges)
    print("Threshold",threshold)
#  Final image after Ptile Method.
    %matplotlib inline
    from matplotlib import pyplot as plt
    plt.imshow(finaloutputarray,cmap="gray")
    import imageio as i
    i.imwrite("Image_ %f.bmp"%name1,finaloutputarray)
    plt.show()
if __name__ == '__main__':
    main()
```