Computer Vision - Canny Edge Detector Project

Sarjak Pankaj Thakkar - SPT308 - N11078382

Source File Name :

File Type - `Python3`

File Name - `SPT308-CV.py`

Instructions on running the code :

- Libraries Required
  1. Numpy - for matrix operations
  2. Python Imaging Library (PIL) - for loading and saving image
  3. Matplotlib - for viewing the processed image
  4. ArgParse - for passing arguments from the terminal

- How to run the program
  1. Installing required libraries
     ```
     sudo pip install numpy pillow matplotlib
     ```
  2. Running the code
     ```
     python SPT308-CV.py --imagePath "<path-to-image>" --ptilingprop <num-edges-in-percentage>
     ```
  3. For help regarding running the program
     ```
     python SPT308-CV.py --help
     ```

Processed Images

1. *Lena256.bmp*

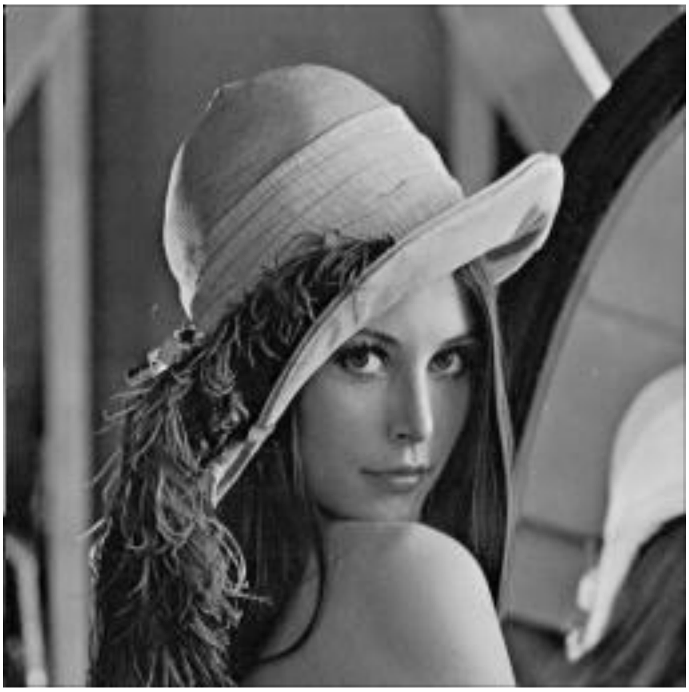
```
python SPT308-CV.py --imagePath "Lena256.bmp" --ptilingprop 10
```

```
python SPT308-CV.py --imagePath "Lena256.bmp" --ptilingprop 30
```

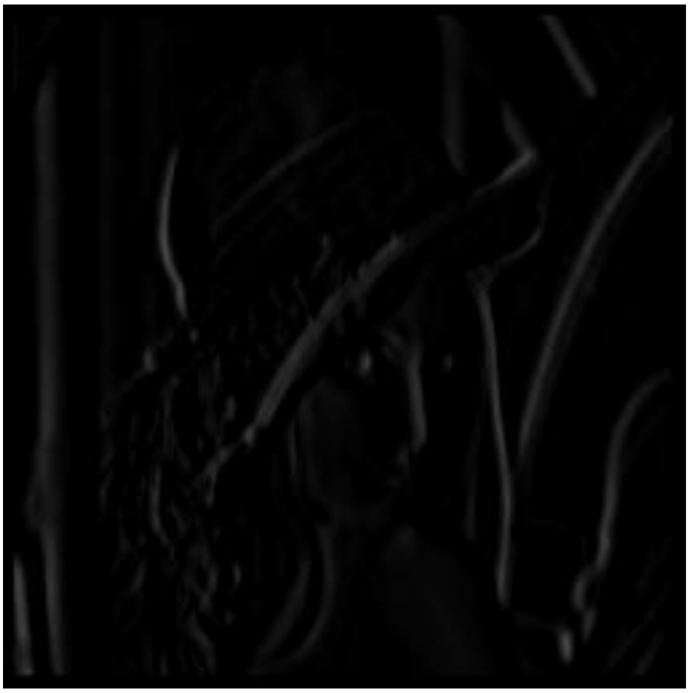
```
python SPT308-CV.py --imagePath "Lena256.bmp" --ptilingprop 50
```

1. Original Image

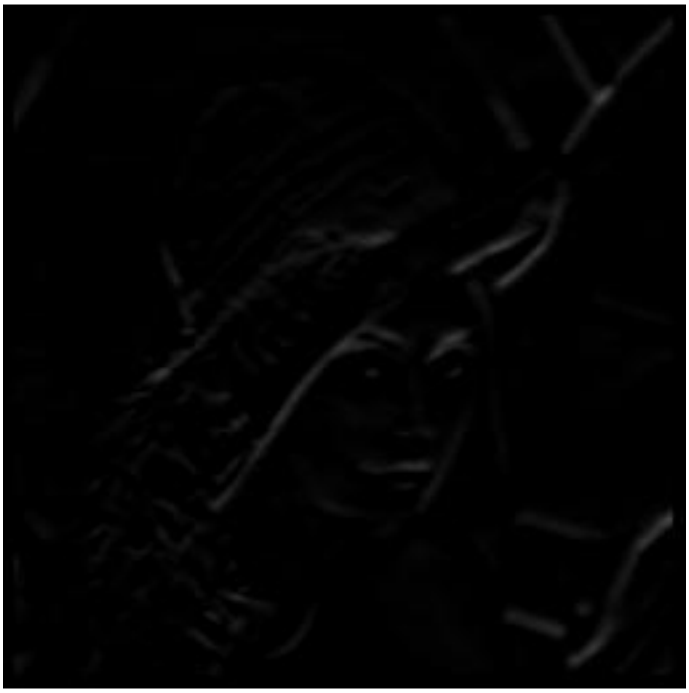
2. Image processed after Gaussian Filter



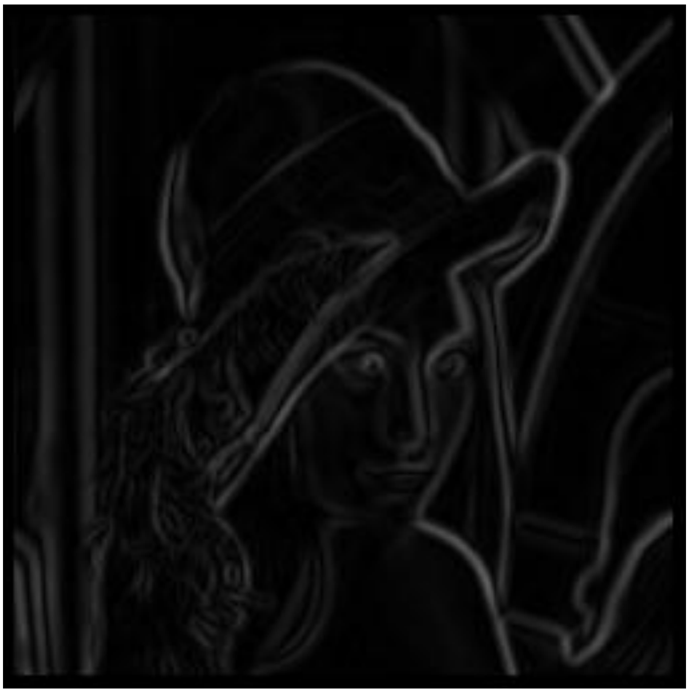
3. Gradient Image along X - Axis



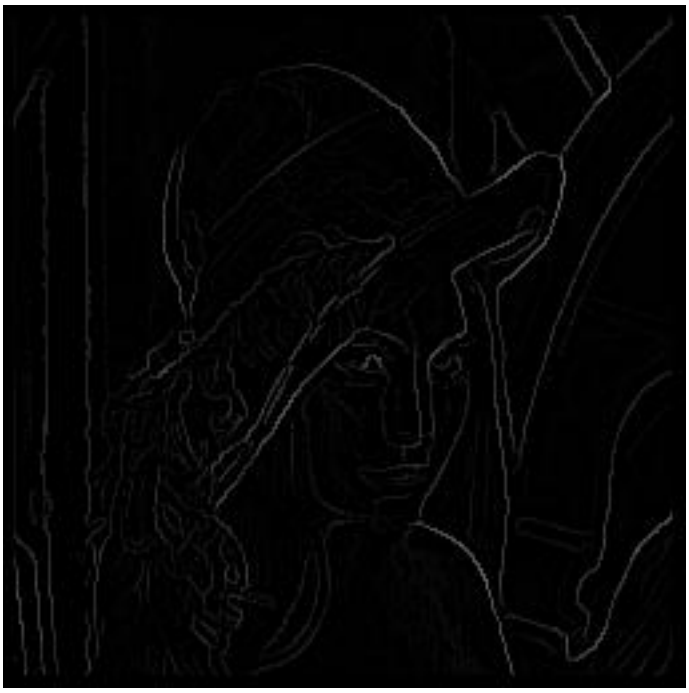
4. Gradient Image along Y-Axis



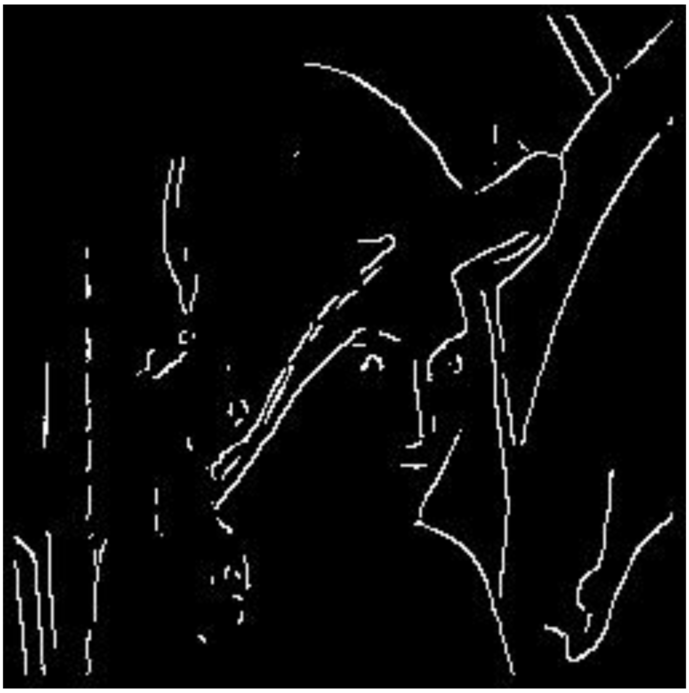
5. Gradient Magnitude Image



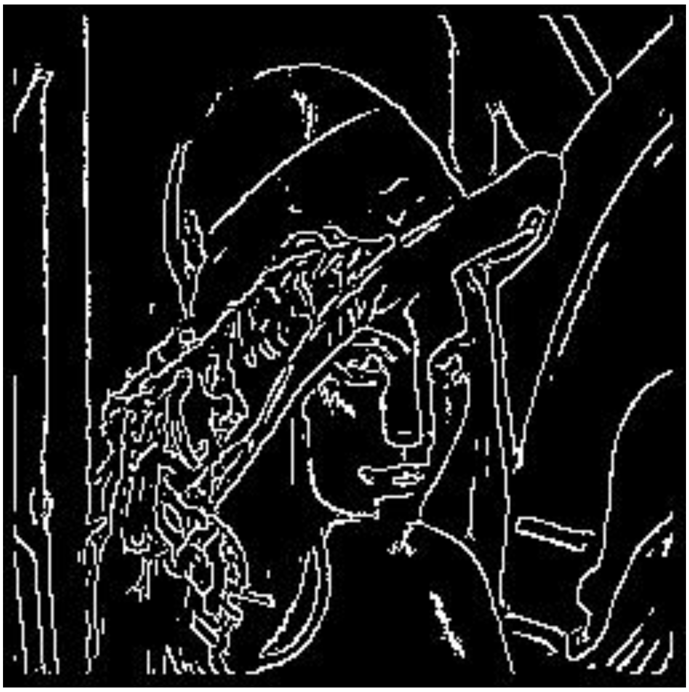
6. Non-Maxima Suppressed Image



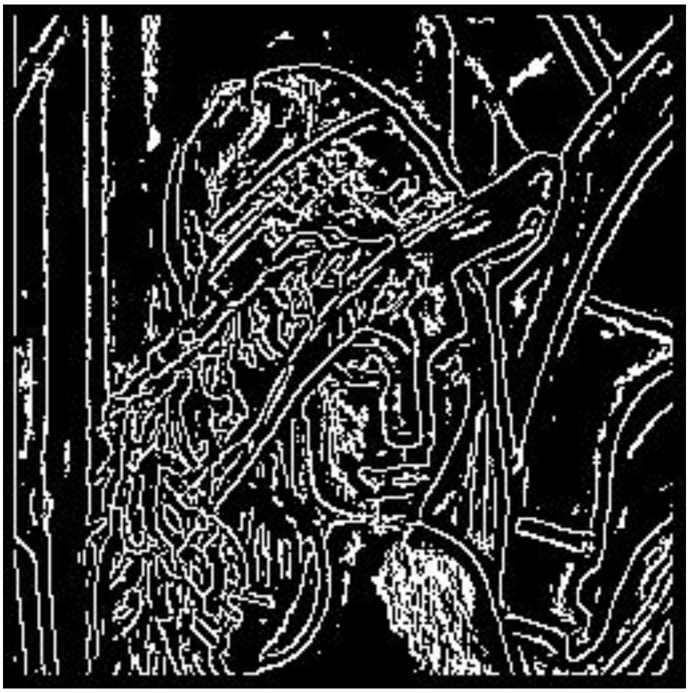
7. Image after P-Tiling Thresholding at 10%
    - Threshold Pixel Value = 34
    - Number of Pixels in Edge = 2213



8. Image after P-Tiling Thresholding at 30%
    - Thresholding Pixel Value = 10
    - Number of Pixels in Edge = 6673



9. Image P-Tiling Thresholding at 50%
    - Thresholding Pixel Value = 4
    - Number of Pixels in Edge = 12628



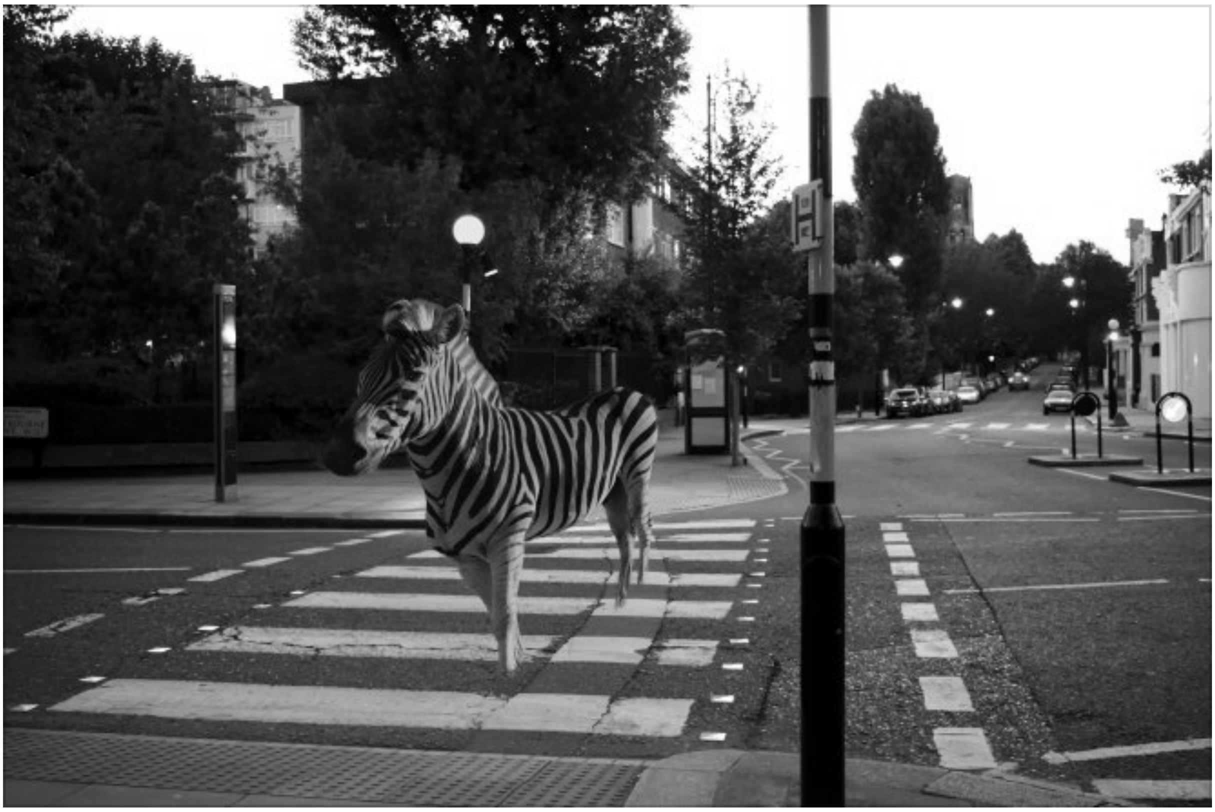
2. *zebra-crossing-1.bmp*

```
python SPT308-CV.py --imagePath "zebra-crossing-1.bmp" --ptilingprop 10
```

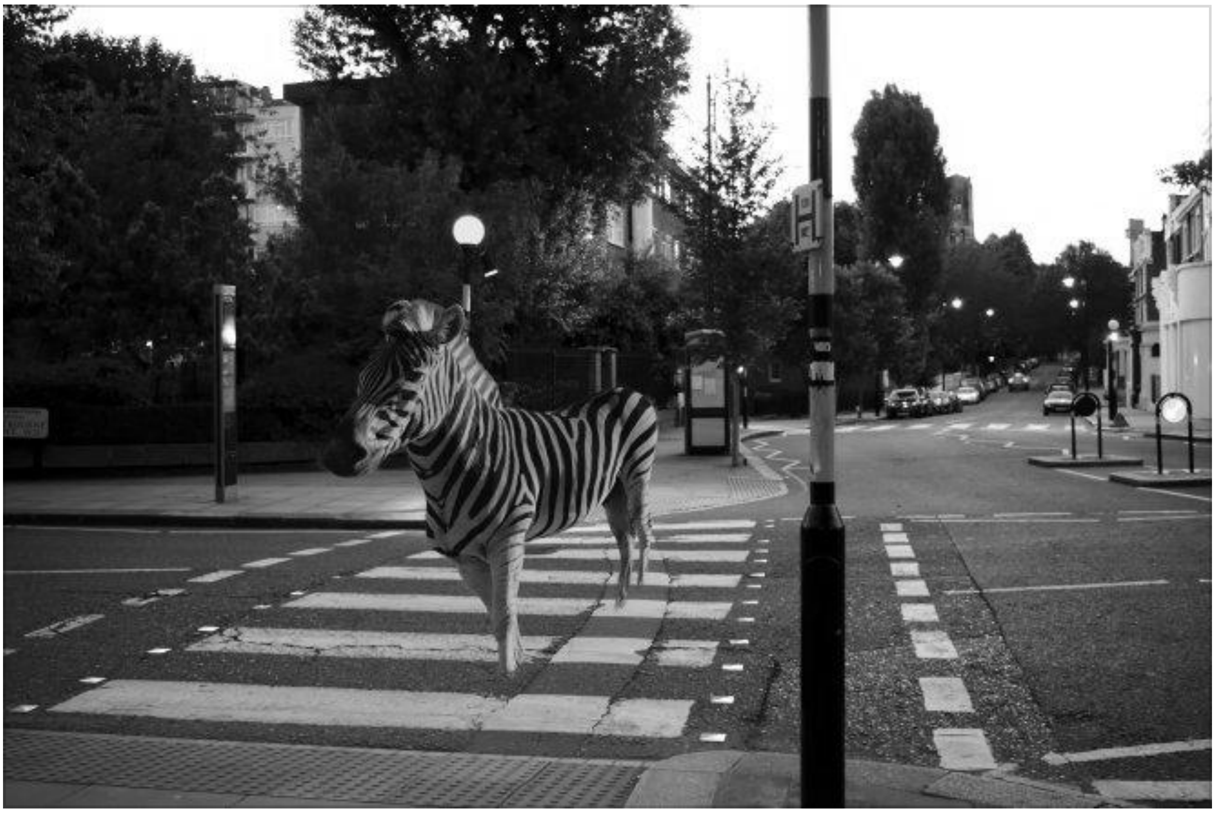
```
python SPT308-CV.py --imagePath "zebra-crossing-1.bmp" --ptilingprop 30
```

```
python SPT308-CV.py --imagePath "zebra-crossing-1.bmp" --ptilingprop 50
```

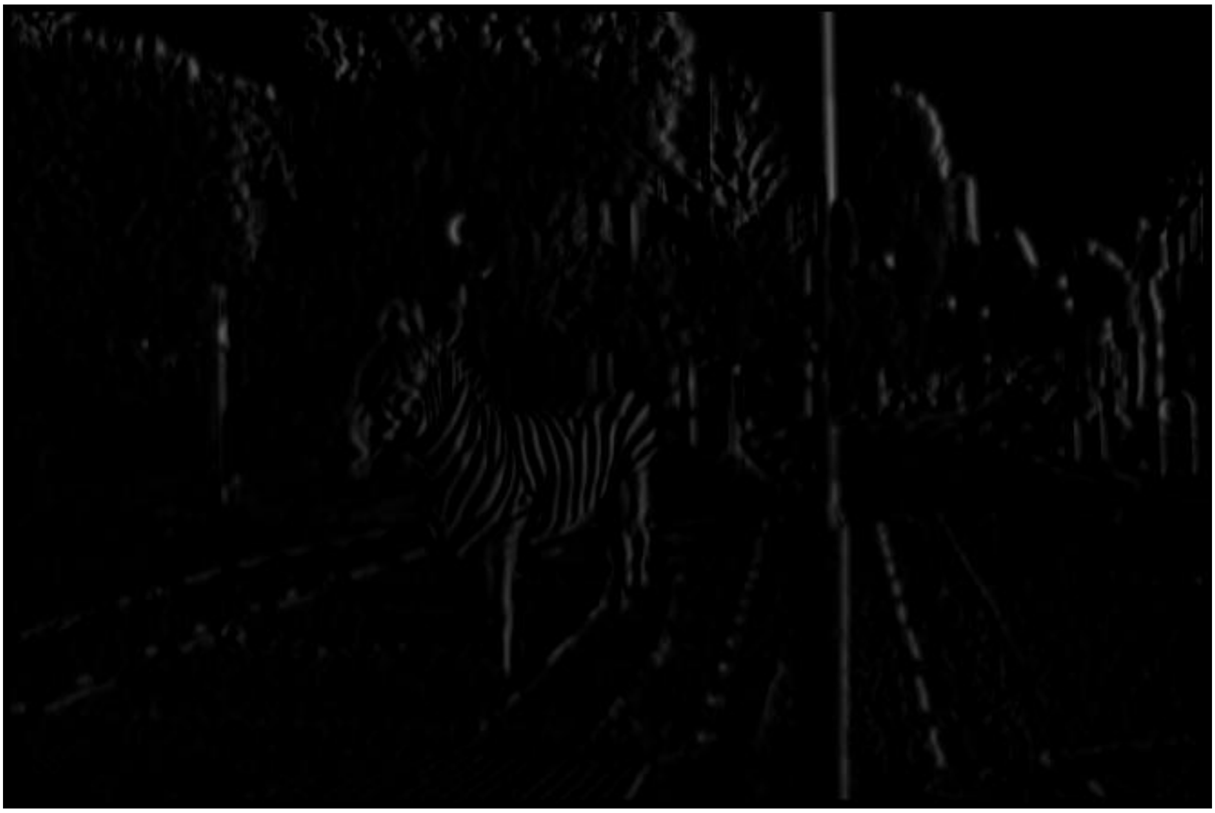
   1. Original Image



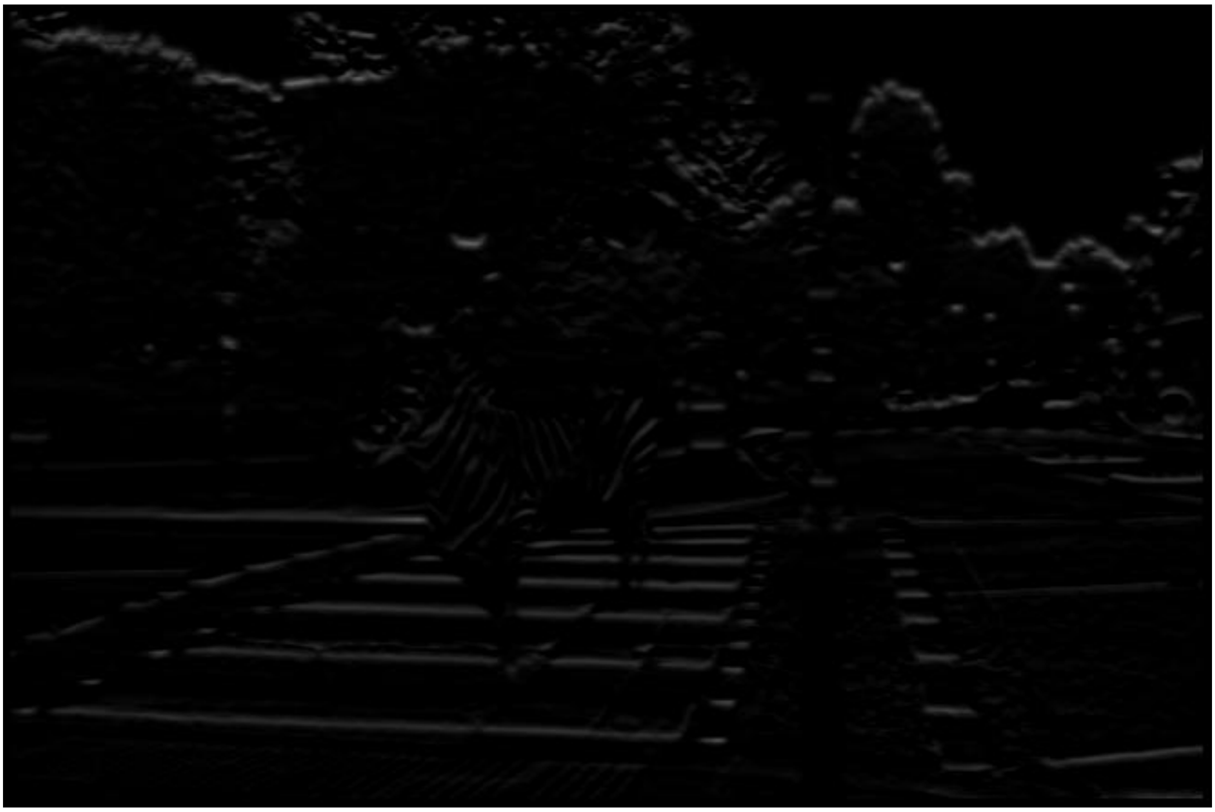
2. Image processed after Gaussian Filter



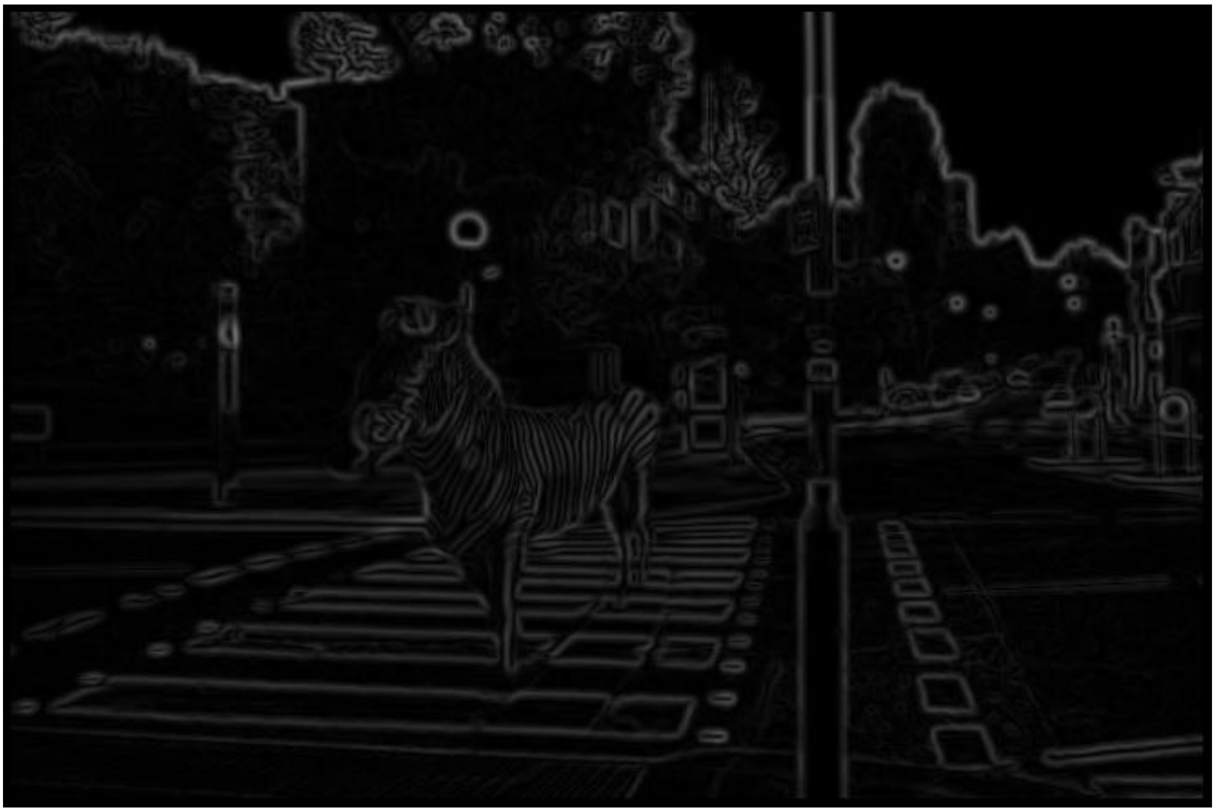
3. Gradient Image along X - Axis



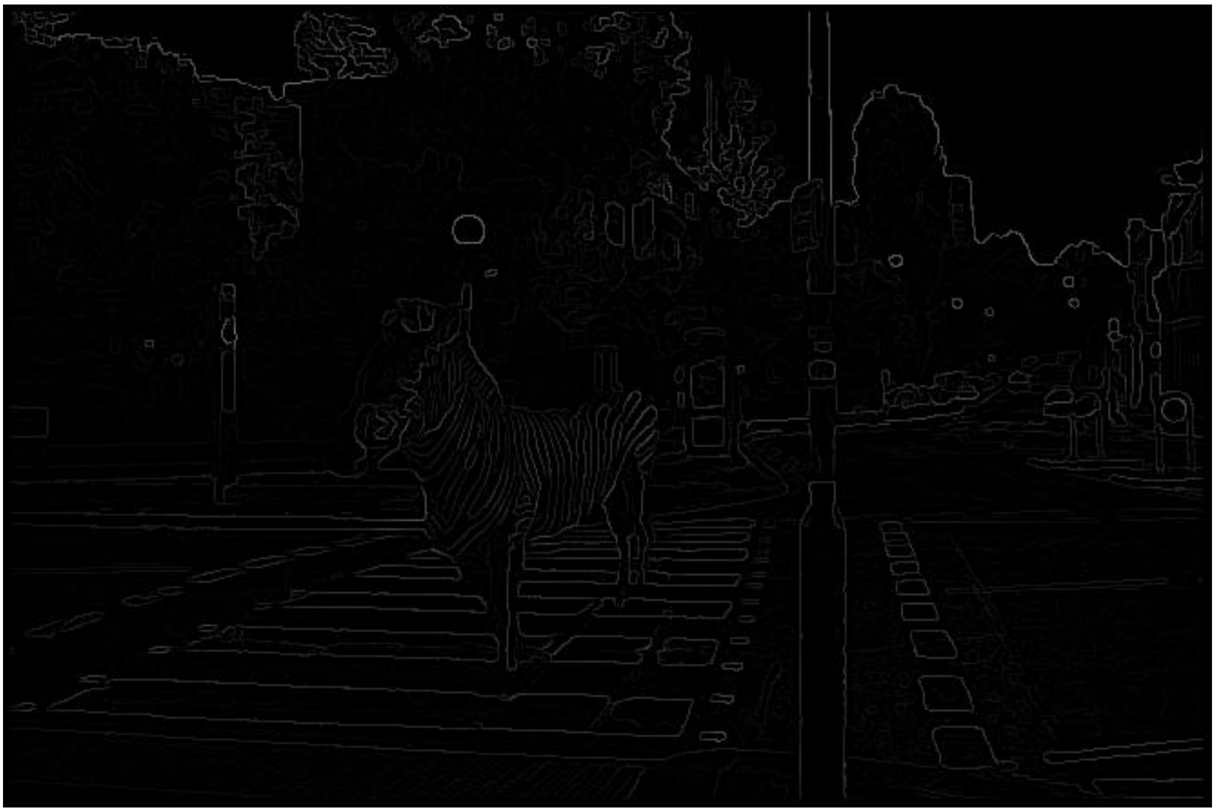
4. Gradient Image along Y-Axis



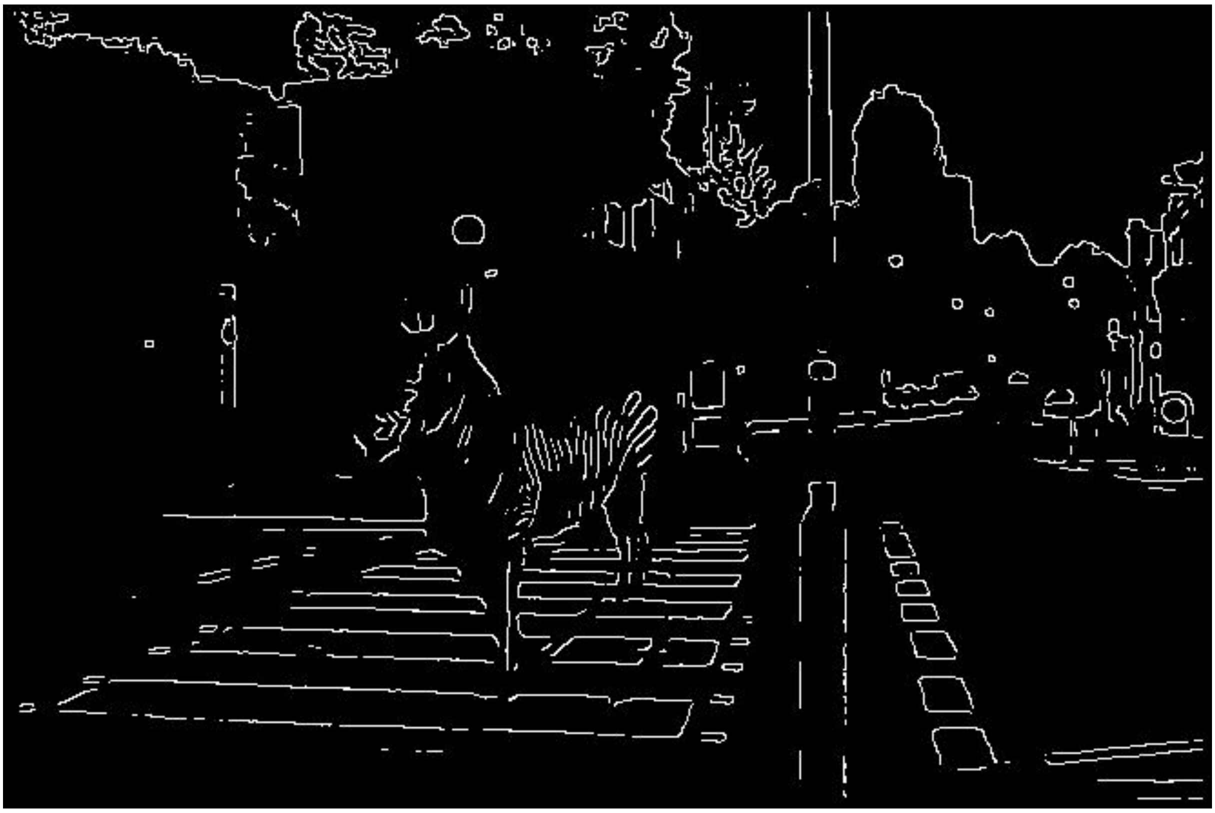
5. Gradient Magnitude Image



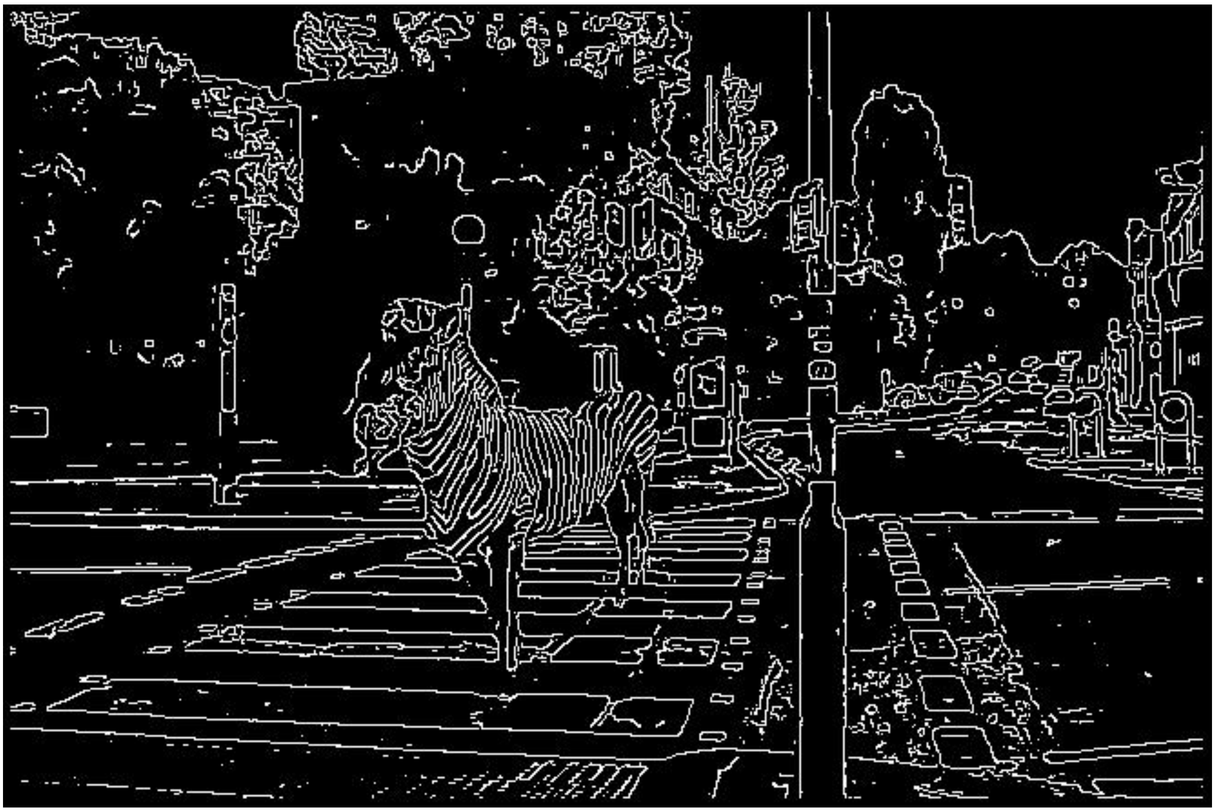
6. Non-Maxima Suppressed Image



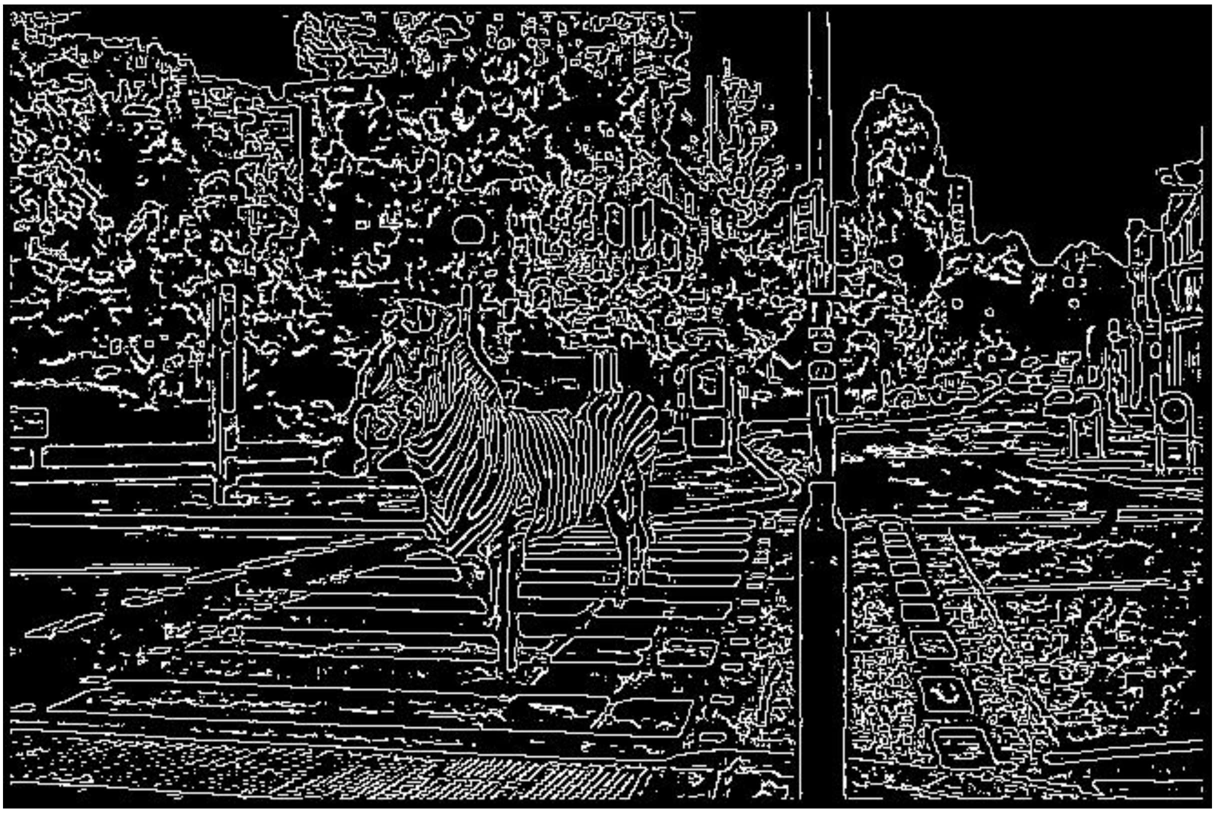
7. Image after P-Tiling Thresholding at 10%
   - Threshold Pixel Value = 36
   - Number of Pixels in Edge = 9820



8. Image after P-Tiling Thresholding at 30%
    - Thresholding Pixel Value = 9
    - Number of Pixels in Edge = 29350



9. Image P-Tiling Thresholding at 50%
    - Thresholding Pixel Value = 4
    - Number of Pixels in Edge = 53536



Source Code :

```
# Numpy for matrix operations
import numpy as np

# PIL for reading and saving image
from PIL import Image

# matplotlib for displaying the processed image
import matplotlib.pyplot as plt

# For passing arguments from command-line
import argparse

# To Ignore the warnings of NaN/NaN calculations by numpy
np.warnings.filterwarnings('ignore')

path = ''
prop = 30
```

```
def parse_args():
  '''
  Reads the commands from terminal for image path and ptiling proportion
  '''

  # Desription for the code
  parser = argparse.ArgumentParser(description='Canny Edge Detector Program for Computer Vision Project - Sarjak Pankaj Thakkar - SPT308')

  # Adding the Argument to read the path of the image
  parser.add_argument('--imagePath', type=str, help='Path of your image to be processed')

  # Adding the Argument to read the ptiling proportion for processing
  parser.add_argument('--ptilingprop', type=float, help='Ptiling proportion to be used for edges')

  # Storing all the read arguments into a variable to return
  args = parser.parse_args()

  # Return all the read arguments
  return args

def readImage(path):
  '''
  Reads an image and return an array containing pixel intensity values
  input   : type - String
            value - Path to the image

  Returns : type - 2D numpy array
            value - Pixel Intensity Values at each pixel
  '''

  # Open the image as a PIL image in grayscale
  img = Image.open(path).convert('L')

  # Convert the PIL image to a numpy array
  img = np.array(img)
  # Return the numpy array
  return img

def applyGaussian(img):
  '''
  Reads an array of of pixel intensity values and applies 7x7 gaussian
smoothening

  input   : type - numpy array
            value - Pixel Intensity values

  Returns : type - numpy array
            values - Pixel Intensity values after gaussian smoothening
  '''

  # Declaring the convolutional mask
  convMask = 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]])

  # Declaring the output image for writing the smoothened values
  output = np.zeros((img.shape[0] - 6, img.shape[1] - 6)) #-6 is correct

  #img2 = img[0:0+7,0:0+7]
  #print(img2.shape)

  # Loop through every pixel
  for x in range(img.shape[0]-6):
      for y in range(img.shape[1]-6):

          # Applying the convolution filter
          output[x,y]=int(((convMask*img[x:x+7,y:y+7]).sum())/140)

  # Assigning the calculated pixel values to outImg
  outImg = np.zeros((img.shape[0], img.shape[1]))
  outImg[3:-3, 3:-3] = output

  # Return the numpy array of image
  return outImg
```

```
def applyGradient(img):
  '''
  Reads an array of of pixel intensity values and applies prewitt's gradient
operator

  input   : tyoe - numpy array
            value - Pixel Intensity values

  Returns : type - two numpy arrays
            values - return 1 : Gradient about X-axis
                     return 2 :  Gradient about Y-axis
  '''

  # Declaring convolutional masks for prewitt's operator
  gradX = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
  gradY = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])

  # Declaring arrays for storing calculated values after prewitt's operator
  outputX = np.zeros((img.shape[0] - 8, img.shape[1] - 8))
  outputY = np.zeros((img.shape[0] - 8, img.shape[1] - 8))

  # Loop over every pixel of the image
  for x in range(3, img.shape[0]-6):     # 3-(-6) is correct
      for y in range(3, img.shape[1]-6):
          outputX[x-3,y-3]=((gradX*img[x:x+3,y:y+3]).sum())
          outputY[x-3,y-3]=((gradY*img[x:x+3,y:y+3]).sum())

  # Normalizing the image values (setting the range to [0,255])
  outputX = normalize(outputX)
  outputY = normalize(outputY)

  # Assigning the calculated pixel values to outImg
  outImgX = np.zeros((img.shape[0], img.shape[1]))
  outImgY = np.zeros((img.shape[0], img.shape[1]))
  outImgX[4:-4,4:-4] = outputX
  outImgY[4:-4,4:-4] = outputY

  # Return the numpy arrays of image
  return outImgX, outImgY
```

```
def calculateGradMag(gradX, gradY):
  '''
  Reads two arrays of gradient along X-axis and Y-axis and calculates the
gradient magnitude

  input   : type - two numpy arrays
            value - gradient along X-axis and Y-axis

  Returns : type - numpy array
            values - magnitude of gradient for given gradients
  '''

  # Calculate the magnitude by square root of sum of squares
  magnitude = np.sqrt((gradX * gradX) + (gradY * gradY))

  # Return magnitude
  return magnitude



def normalize(img):
  '''
  Reads an input array and normalises the values to set in range [0,255]

  input   : type - numpy array
            value - pixel values

  Returns : type - numpy array
            values - normalized pixel values
  '''

  img = img/3

  return img

def calcGradAngle(gradX , gradY):
  '''
  Reads two arrays of gradient along X-axis and Y-axis and calculates the
gradient angle at each pixel

  input   : type - two numpy arrays
            value - gradient along X-axis and Y-axis

  Returns : type - numpy array
            values - gradient angle for given gradients
  '''

  # Using arctan (tan inverse) to find angle in radians
  angle = np.arctan(np.true_divide(gradX, gradY))

  # Converting angle in radians to degrees
  angle = angle * (180 / np.pi)

  # Return angles array
  return angle

def nonMaximaSuppression(magnitude, angle):
  '''
  Reads two arrays of gradient along X-axis and Y-axis and calculates the
gradient angle at each pixel

  input   : type - two numpy arrays
            value - gradient magnitude and gradient angle at every pixel

  Returns : type - numpy array
            values - gradient angle for given gradients
  '''

  output = np.zeros((magnitude.shape[0], magnitude.shape[1]))

  for x in range(1, magnitude.shape[0]-1):     # Loop over every pixel of the
image
      for y in range(1, magnitude.shape[1]-1):

          currAngle = angle[x,y] # To decrease the runtime, dont have to look
```

```
up the array for every condition
          currMagnitude = magnitude[x,y] # To decrease the runtime, dont have
to look up the array for every condition

          # No need of else statement (in nested if conditions) as entire
matrix is initially set to zero

          if -22.5 <= currAngle <= 22.5 or currAngle <= -157.5 or currAngle >=
157.5: # Case 0
              if currMagnitude >= np.amax([currMagnitude, magnitude[x+1,y],
magnitude[x-1,y]]):
                  output[x,y] = currMagnitude

          elif 22.5 <= currAngle <= 67.5 or -112.5 >= currAngle >= -157.5: #
Case 1
              if currMagnitude >= np.amax([currMagnitude, magnitude[x+1,y-1],
magnitude[x-1,y+1]]):
                  output[x,y] = currMagnitude

          elif 67.5 <= currAngle <= 112.5 or -67.5 >= currAngle >= -112.5: #
Case 2
              if currMagnitude >= np.amax([currMagnitude, magnitude[x,y-1],
magnitude[x,y+1]]):
                  output[x,y] = currMagnitude

          elif 112.5 <= currAngle <= 157.5 or -22.5 >= currAngle >= -67.5: #
Case 3
              if currMagnitude >= np.amax([currMagnitude, magnitude[x-1,y-1],
magnitude[x+1,y+1]]):
                  output[x,y] = currMagnitude

  # Return the nonmaxima suppresses image
  return output

def generate_histogram(img):
  '''
  Reads numpy array of pixel intensity values and computes a histogram

  input   : type - numpy array (2D)
```

```
            value - pixel intensities of an image

  Returns : type - two numpy array (1D)
            values - histogram values for each pixel intensity value and
                     number of non-zero pixel intensity pixels in the image
  '''

  # Create an empty histogram for future increments
  histogram = np.zeros(360)
  nonZeroPixels = 0

  # Iterating through each pixel
  for x in range(0, img.shape[0]):
      for y in range(0, img.shape[1]):

          # Incrementing suitable index in histogram array
          currIntensity = int(img[x,y])
          if currIntensity != 0:
              nonZeroPixels += 1

          histogram[currIntensity] += 1

  # Return histogram and non-zero pixels
  return histogram, nonZeroPixels

def findThreshold(nonZeroPixels, histogram, proportion):
  '''
  Reads numpy arrays of histogram, number of non-zero pixels and ptiling
proportion

  input   : type - numpy arrays
            value - histogram, number of non-zero pixels and ptiling proportion

  Returns : type - integer
            values - threshold value
  '''

  i = 359
  total = 0
```

```
    breakFlag = False

    # Calculating the number of pixels in foreground
    proportion = proportion * nonZeroPixels

    # Looping through all pixel intensity values in histogram
    while(i > 0 and breakFlag == False):
        value = histogram[i]
        total = total + value

        i = i-1

        # If ptiling proportion exceeded, break
        if total > proportion:
            breakFlag = True

    threshold = i

    # Return Threshold Value
    return threshold

def ptiling(img, threshold):
    '''
    Reads an image and applies thresholding
    input   : type - numpy array
              value - pixel intensity values

    Returns : type - 2D numpy array
              value - thesholded image
    '''

    # Variable to keep count of number of pixels in the edges
    numEdge = 0

    # Loop over every pixel in the image
    for x in range(0, img.shape[0]):
        for y in range(0, img.shape[1]):

            # Check if above or below threshold value and apply thresholding
```

```
          if img[x,y] < threshold:
              img[x,y] = 0
          else :
              img[x,y] = 255
              numEdge += 1

  # Return processed image
  return img, numEdge

def main():

  # Reading the image
  img = readImage(path)

  # Applying Gaussian smoothening to the Image
  imgGaus = applyGaussian(img)
  saveImg = Image.fromarray(img).convert("RGB")
  saveImg.save(path[:-4] + "_Gaussian" + ".jpg")

  # Calculating Gradient-X and Gradient Y for the Image
  gX, gY = applyGradient(imgGaus)
  print("Max Gradient X : ", np.amax(gX))
  saveImg = Image.fromarray(gX).convert("RGB")
  saveImg.save(path[:-4] + "_GradientX" + ".jpg")

  print("Max Gradient Y : ", np.amax(gY))
  saveImg = Image.fromarray(gY).convert("RGB")
  saveImg.save(path[:-4] + "_GradientY" + ".jpg")

  # Calculating the Gradient Magnitude for the image
  magnitude = calculateGradMag(gX, gY)

  # Taking the rounded values of magnitude and ignoring the decimal points
  magnitude = np.around(magnitude)
  saveImg = Image.fromarray(magnitude).convert("RGB")
  saveImg.save(path[:-4] + "_Gradient_Magnitude" + ".jpg")
  print("Max Magnitude : ",np.amax(magnitude))

  # Calculating the Gradient-Angle for each pixel value
  angle = calcGradAngle(gX,gY)

  # Applying non-maxima suppression to the image
  nonMaxima = nonMaximaSuppression(magnitude,angle)
  saveImg = Image.fromarray(nonMaxima).convert("RGB")
  saveImg.save(path[:-4] + "_nonMaximaSuppressed" + ".jpg")

  # Calculating the histogram and the number of non-zero pixels in the image
  histogram, nonZeroPixels = generate_histogram(nonMaxima)

  # Finding the threshold using the ptiling method
  threshold = findThreshold(nonZeroPixels, histogram, prop/100)
  print("Threshold value : ", threshold)

  # Applying the theshold for the image based on ptiling
  edgedimg, numEdges = ptiling(nonMaxima, threshold)
  print("Number of pixels in edge of the image ", numEdges)

  # Saving the image
  saveImg = Image.fromarray(edgedimg).convert("RGB")
  saveImg.save(path[:-4] + "_processed_" + str(prop) + "%_Threshold=" + str(threshold) + ".jpg")

  # Displaying the processed image
  plt.imshow(edgedimg, cmap='gray')
  plt.show()

if __name__ == '__main__':

  # Reading the arguments from the Argument parser function
  user_choice = parse_args()

  # If no filepath of image given, return error. Else assign to path
  if not user_choice.imagePath:
      print('Error - No filename entered')
  else:
      path =  user_choice.imagePath
      print("Opening image at ", path)
```

~~~
      # If no ptiling proportion given, set the default proportion to 30%. Else, assign to specified
      if not user_choice.ptilingprop:
          print("Using default Ptiling Proporation of 30%")
      else:
          prop = user_choice.ptilingprop
          print("Using ptiling proportion ", prop)

      # Call the main function
      main()

```
~~~