Project II: Canny Edge Detector

1 Instructions

- This project is due on September 21.
- Besides an ipynb file, you need to submit a pdf report with edge detection results for images under "Images" and "Test_Images" in it.
- This is a group project. You can work with another team mate on this project. Both collaborators should make a submission and clearly indicate who they are coordinating with.
- You must submit your code and report on Gradescope.

2 Introduction

In this project, we will implement the Canny edge detector. The project focuses on understanding the image convolution and edge detection. As summarized in our course material, Canny edge detector is implemented via,

- Filter image by derivatives of Gaussian
- Compute magnitude of gradient
- Compute edge orientation
- Detect local maximum
- Edge linking

We follow the process to implement a function for edge detection. The final goal of this project is to compute the Canny Edges for any RGB image as shown in Figure 1.

Figure 1: Input Image and Edge Map

E = cannyEdge(I)

- (INPUT) **I**: H x W x 3 matrix representing the RGB image, where H, W are the height and width of the input image respectfully.
- (OUTPUT) **E**: H x W binary matrix representing the canny edge map, where a 1 is an Edge pixel while 0 is a Non-Edge pixel.

Your task is to implement two basic functions findDerivatives nonMaxSup which correspond to the steps described in the lecture notes:

1. Apply Gaussian smoothing and compute local edge gradient magnitude as well as orientation.
2. Seek local maximum edge pixel in corresponding orientation.
3. Find the appropriate thresholds to get good edges

3 Image Derivative

The filters for gradient along x- and y-axis are as follows.

$$d_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad d_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Remember y axis goes down and d_x, d_y will be flipped when doing convolution. The direct use of the filters will give the noisy gradient maps that influence the subsequent edge detection. To suppress noises, Gaussian smoothing is usually adopted.

$$Gaussian = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

In function **findDerivatives**, you will implement Gaussian smoothing, compute magnitude and orientation of derivatives for the image. The convolution settings

used should be "same" with padding values of 0. The protocol for the function is summarized as follows:
[Mag, Magx, Magy, Ori] = findDerivaties(I_gray)

- (INPUT) **I_gray**: H x W matrix representing the grayscale image.
- (OUTPUT) **Mag**: H x W matrix representing the magnitude of derivatives.
- (OUTPUT) **Magx**: H x W matrix representing the magnitude of derivatives along the x-axis.
- (OUTPUT) **Magy**: H x W matrix representing the magnitude of derivatives along the y-axis.
- (OUTPUT) **Ori**: H x W matrix representing the orientation of derivatives.

4 Detect Local Maximum

The function 'nonMaxSup' is to find local maximum edge pixel using non-maximum suppression along the line of the gradient. The operation further suppresses noises. You will implement the function whose protocol is as follows.
M = nonMaxSup(Mag, Ori)

- (INPUT) **Mag**: H x W matrix representing the magnitude of derivatives.
- (INPUT) **Ori**: H x W matrix representing the orientation of derivatives.
- (OUTPUT) **M**: H x W binary matrix representing the edge map after non-maximum suppression.

5 Edge Linking

After non maximum suppression, we have the potential edge map. To get the final result, we use low and high thresholds to divide current edges to three categories. If the Magnitude of the edge is below low threshold, then it is noise and we discard it. If it is above high threshold, then we accept it as part of final edge map. We also call it the strong edge. For these edges that are between two thresholds, we are uncertain about them. They are also known as weak edges. In the **edgeLink** function, we try to link weak edges to strong edges. Those weak edges that can be connected to strong edges stay in the final edge map and others are discarded. The protocol is summarized below:
E = edgeLink(M, Mag, Ori, low, high)

- (INPUT) **M**: H x W logical map after non-max suppression
- (INPUT) **Mag**: H x W matrix represents the magnitude of gradient

- (INPUT) **Ori**: H x W matrix represents the orientation of gradient
- (INPUT) **low**: low threshold
- (INPUT) **high**: high threshold
- (OUTPUT) **E**: H x W binary matrix represents the final canny edge detection map

6 Provided Functions

- (Function) **v2 = interp2(v, xq, yq)**
  - (INPUT) **v**: H x W value to be interpolated
  - (INPUT) **x**: N x 1, x coordinated of the interpolation
  - (INPUT) **y**: N x 1, y coordinated of the interpolation
  - (OUTPUT) **v2**: N x 1, output. v2 = v[x, y] where x, y can be any real values within v instead of just integer values.

7 Threshold tuning

For each image, we need to tune low and high thresholds and keep the result in the `thresh_dict`. The rule of thumb is to tune the high threshold so that enough edges are preserved and then setting low threshold to be around 0.4 high threshold to remove noise. Don't trust the final edge detection result in the notebook since it is too small and many details are lost. Check out the saved image instead to see the real result.

8 Rubrics

Passing tests for three functions **findDerivaties**, **nonMaxSup**, **edgeLink**	30 pts
Get good edge result for the checkerboard image	20 pts
Get good edge result for the coin image	20 pts
Get good edge results for images under Images folder	30 pts