Project 2 - Canny Edge Detector

1 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.

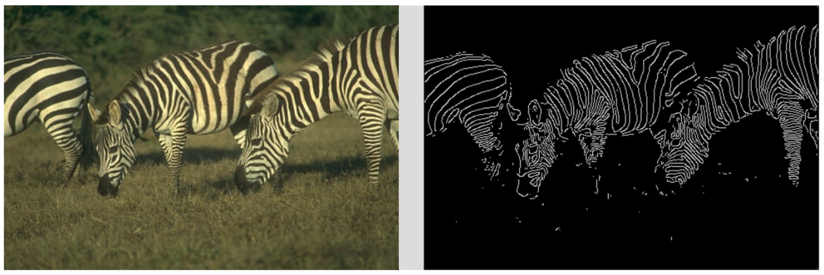

Figure 1: Input image and edge map

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()` and `nonMaxSup()` which correspond to the first two steps described in the lecture notes:

1. Apply Gaussian smoothing and compute local edge gradient magnitude as well as orientation.
2. Perform Non-Maximum Suppression to find local maximum edge pixel in corresponding orientation.
3. Perform edge linking to find the appropriate thresholds to get good edges.

2 Image derivative

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

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

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.

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

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] = findDerivatives(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.

3 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.

4 Provided functions

The helper function `interp2()` is provided for you to get the interpolated values, which is useful to find neighboring pixel values.

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.

The helper function `edgeLink()` is provided for you to perform edge linking hysteresis check such that only strong edges and those weak edges connecting to strong edges are preserved in the final edge map.

E = edgeLink(M, Mag, edge_Ori, low, high)

- (INPUT) **M**: H x W binary matrix representing the edge map after non-maximum suppression.
- (INPUT) **Mag**: H x W matrix representing the magnitude of derivatives.
- (INPUT) **edge_Ori**: H x W matrix representing the discrete orientation of edges.
- (INPUT) **low**: low end of the magnitude value for determining if an edge belongs in the final edge map.
- (INPUT) **high**: the value used to determine strong edges
- (OUTPUT) **E**: H x W binary matrix representing the final Canny Edge map.

5 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.

6 Submission

Please submit the following files:

1. Your completed .ipynb file. Please rename your .ipynb file in the following manner with your name: CIS5810_22Fall_Project2_FirstName_LastName
2. The edge result image under the *Results* folder. There should be **14** edge result images in total, including
   118035_Result.png, 135069_Result.png, 16068_Result.png, 189080_Result.png, 201080_Result.png, 21077_Result.png, 22013_Result.png, 3096_Result.png, 48017_Result.png, 55067_Result.png, 86000_Result.png, coins_Result.png, I1_Result.png, rotated_checkerboard_Result.png