EECS 4422/5323 Computer Vision
Assignment 2: Image Filtering

All programming questions are to be completed in Python. For each individual programming question, submit a Python script. Include a file, call it `a1_script.ipynb`, that will step through the answers to the entire assignment with each question appropriately sectioned by notebook cells. Note that you can select the 'cell type' and add large comments in between cells.

In this assignment, you should use the `Kornia Library` and PyTorch instead of NumPy. You may find `the following Kornia tutorials page useful.` For those of you not working on York's lab computers, `pip install kornia` should install all the necessary libraries. PyTorch uses almost the exact same syntax as NumPy (e.g., tensors will multiply along intuitive axis and for-loops should be avoided when possible, i.e., vectorize your code). Note that most PyTorch (and Kornia) functions require the shape of the array to be $B \times C \times H \times W$, where B represents a 'batch' of images (this will be 1 in your case), C is the channels, and H and W are height and width, respectively. Also, make sure you cast your array types to the proper type (e.g., `torch.float32` in most cases) before performing operations on them. Matplotlib only works with NumPy arrays, so make sure you change your PyTorch tensors back to NumPy arrays before displaying. You may find the functions in `the following PyTorch tutorial webpage useful.`

Unless otherwise specified, all image processing steps are to be performed on greyscale images (if images are colour, use `kornia.rgb_to_grayscale()`). Finally, **do not share code or use code from the web or any other source**. We will be checking for copying using the MOSS system. Offenders (both copier and source) will receive a zero on this assignment and we may pursue academic sanctions. **This assignment is to be done individually.**

1 Canny edge detection (Total 45)

1. [**30 Points**] Implement the Canny edge detection algorithm (up to but not including the hysteresis step) as a Python function, call it `MyCanny`, as described in class and the handout authored by Pedro Felzenszwalb. Your function should take as input a greyscale image and the edge detection parameters and return the Canny edge binary image. For this question, there are two edge detection parameters, the standard deviation, σ, of the Gaussian smoothing filter and the gradient magnitude threshold, τ. Note, if you implement the peak/ridge detection step correctly the output of your program should NOT have "thick" edges!

   In addition to turning in your source code for the Canny edge detector, submit a Python script that runs your edge detector on the test image provided at this link and on an image of your own choosing. Your Canny output images should use the best set of parameters you find for each input image.

   Note: You are excluded from using Kornia's `SpatialGradient(mode='sobel')` function in your own code but are encouraged to run the `SpatialGradient(mode='sobel')` function to get a sense of what the correct output should look like.

2. [**10 Points**] Extend the Canny function you wrote for Q1.1 to include the hysteresis mechanism, call it `MyCannyFull`. For hysteresis thresholding, a high and low threshold is specified by the user beforehand. The process begins by marking all pixels with gradient magnitudes above the high threshold as a "discovered" definite edge. These pixels are placed into a queue and become the starting points for a breadth first search (BFS) algorithm. Run the BFS by iterating through the queue of pixels. The hysteresis process terminates when the queue is empty. All adjacent pixels (one of the eight neighbours) are treated as connected nodes to the current pixel removed from the queue. The criteria for adding a new pixel to the queue is given by: an adjacent pixel that has not been previously discovered and has a gradient magnitude greater than the low threshold. Adjacent pixels that meet the criteria are subsequently added to the BFS queue. Every adjacent pixel is also marked as discovered once it is checked against the criteria.

2 Seam carving (Total 20 points)

1. **[20 Points]** Seam carving is a procedure to resize images in a manner that preserves "important" image content. A video demo is available on YouTube. The general steps for seam carving are as follows:

   (a) Compute the energy image, E, for the input image, e.g., the sum of the gradient magnitude images computed for each of the three colour channels of the input image. Here, you can use Kornia's `SpatialGradient(mode='sobel')`.

   (b) Create a scoring matrix, M, with spatial image dimensions matching those of the input image.

   (c) Set the values of the first row of the scoring matrix, M, to match those of the energy image, E.

   (d) Set the values of every entry in the scoring matrix to the energy value at that position and the minimum value in any of the neighbouring cells above it in the seam, i.e.,

   $$M[\text{row}, \text{col}] = E[\text{row}, \text{col}] + \min\Big(M[\text{row}-1, \text{col}-1], M[\text{row}-1, \text{col}], M[\text{row}-1, \text{col}+1]\Big), \tag{1}$$

   where $M[\text{row}, \text{col}]$ represents the lowest accumulated cost among all seams terminating at that point. This minimization procedure is an instance of *dynamic programming*.

   (e) Find the minimum value in the bottom row of the scoring matrix. The corresponding position of the minimal value is the bottom of the optimal seam.

   (f) Using $M[\text{row}, \text{col}]$, trace back up the seam by following the smallest value in any of the neighbouring positions above.

   (g) Remove the seam from the image.

   (h) To reach a desired resized image, you will have to repeat the above procedure. Note that you will have to recompute the energy matrix (and scoring matrix) each time to take into account changes in the resized image.

   Your task is to write a Python function, call it `MySeamCarving`, that takes in an image and the desired new resolution by removing the necessary horizontal and vertical seams and returns the resized image. Implement the seam carving routine with a *single* helper function, call it `CarvingHelper`, that removes all the necessary vertical (horizontal) seams. You first call the helper function to remove the vertical (horizontal) seams, transpose the result of this function and then call the seam removal function again with the transposed image as the input to remove the horizontal (vertical) seams. In addition to submitting your code, submit resizing outputs for the 1728×1151 `York` image to 1200×1151 and 1728×720. Also submit an image of your own and its resized output.

   (a) In addition, implement image expansion by inserting seams, see the original paper for details.