CPS843/CP8307 Introduction to Computer Vision, Winter 2021: Assignment 1 - Filtering
Due: February 23th, 11:59am

All programming questions are to be completed in MATLAB. For each individual programming question, submit a MATLAB script. Include a file, call it `a1_script.m`, that will step through the answers to the entire assignment using `pause` between each question. Unless otherwise specified, all image processing steps are to be performed on greyscale images (if images are colour, use `rgb2gray()`). Finally, *do not share code or use code from the web*. 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 for both CPS843 and CP8307.**

1 Convolution (Total 25)

1. **[5 points]** Fill in the empty table (below-right) with the resulting image obtained after convolution of the original image (bottom-left image) with the following approximation of the derivative filter $[1, 0, -1]$ in the horizontal direction. Assume that the image is zero padded. The origin is located at the top-left corner with coordinates $[0, 0]$. This question is to be *done by hand*; use `fprintf()` in MATLAB to output your response.

-5	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	-7	**2**	1	1	3	0	0	0
0	0	0	1	1	1	1	0	0	0
0	0	0	**3**	1	1	**5**	0	0	0
0	0	0	-1	-1	-1	-1	0	0	0
0	0	0	1	2	3	4	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

2. **[5 points]** Compute the gradient magnitude at pixels $[2, 3]$, $[4, 3]$ and $[4, 6]$ in the left image in Q1.1 (the image pixels marked in bold). Hint: Assume the same derivative filter approximation used for the horizontal direction is used for the vertical direction. Use `fprintf()` in MATLAB to output your response.

3. **[5 points]** Write a convolution function, call it `MyConv`, that takes in as input an arbitrary image and kernel. Your convolution function should only use loops (i.e., do not use any of the prebuilt MATLAB functions, e.g., `imfilter`). For the boundary cases, assume that the values outside the image are zero. (*HINT: Pad the image prior to convolution with zeros.*) Your code should appropriately flip the input kernel.

4. **[5 points]** Compare the output of your convolution function with the output of `imfilter` using a 2D Gaussian kernel with standard deviation 2 and dimensions 13×13. Specifically, subtract the convolution outputs and display the absolute value using reasonable scalings for `imshow`. Is there any difference between the outputs? Make sure that `imfilter` is assuming that the boundaries are zero outside the image. Use `fprintf()` in MATLAB to output your response.

5. **[5 points]** Determine the execution times between convolving a 640×480 image by a 2D Gaussian with a standard deviation of 8 and separably convolving the same image with two 1D Gaussians each with a standard deviation of 8. Use `imfilter` to perform the convolution and `fspecial` to generate the kernels (with the "three-sigma rule"). What do you observe? In MATLAB, if you place the command `tic` before your script and `toc` after your script, MATLAB will return the total execution time. Use `fprintf()` in MATLAB to output your response.

2 Canny edge detection (Total 35 for CPS843 and 45 for CP8307)

1. **[30 points]** Implement the Canny edge detection algorithm as a MATLAB function, call it `MyCanny`, up to but not including the hysteresis step, as described in class and the handout available on the course webpage. 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 MATLAB 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.

   For obvious reasons, you are excluded from using MATLAB's `edge()` function in your own code but are encouraged to run the `edge()` function to get a sense of what the correct output should look like.

2. **[5 points]** Implement the Gaussian convolution as a sequence of horizontal and vertical convolutions, i.e., a separable filter.

3. **[10 points]** (CP8307 question, bonus for CPS843) Add the hysteresis mechanism to the function you wrote for Q2.1. 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.

3 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.

   (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.,

$$M(x, y) = E(x, y) + \min\Big(M(x-1, y-1), M(x, y-1), M(x+1, y-1)\Big), \tag{1}$$

where $M(x, y)$ is the cost of the lowest cost seam crossing through 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 $M(x, y)$, 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 MATLAB 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 `Ryerson` image to 640×480 and 720×320. Also submit an image of your own and its resized output.

2. **[10 points]** (Bonus): Implement image expansion by inserting seams, see the original paper for details.