CPS843/CP8307 Introduction to Computer Vision, Winter 2019: Assignment 1 - Image Filtering Due: Wed., March 1st, 11:59pm

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()`). **This assignment is to be completed in groups of at most two for CPS843 students and individually for CPS8307 students**. For those that choose to work in groups, submit as follows: only one member submits the assignment and in `a1_script.m` print to the screen the names of the group members and their respective student numbers. Finally, *do not share code with those outside your group 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.

1 Convolution in the Spatial Domain (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 Comparing Convolution in the Spatial and Frequency Domain (15 points)

In MATLAB, if you place the command `tic` before your script and `toc` after your script, MATLAB will return the total execution time.

1. **[5 points]** Use `conv2` to convolve an image by a 2D Gaussian kernel in the spatial domain. Plot the execution time to perform convolution on a 640×480 image using the following dimensions for the Gaussian kernel: 3×3, 5×5, 7×7, 13×13, 21×21, 31×31, 41×41, 51×51 and 71×71. Use MATLAB's `plot` function to plot the kernel size (horizontal dimension) vs. the execution time (vertical dimension).

2. **[5 points]** Perform convolution in the frequency domain (i.e., DFT, multiply and inverse DFT), plot the time taken to convolve a 640×480 image with a 2D Gaussian kernel with the following dimensions: 3×3, 5×5, 7×7, 13×13, 21×21, 31×31, 41×41, 51×51 and 71×71. Overlay this plot onto the spatial convolution plot from the previous question using MATLAB's `hold on` function.

3. **[5 points]** Describe the plots you just generated. What conclusions can you draw between spatial- and frequency-based convolution?

3 Hybrid images- Implementation in the Frequency Domain (20 points)

1. **[20 points]** For this question you will create a *Hybrid image* as described in the lecture, e.g., Marilyn Monroe + Albert Einstein. The images used to generate this illusion are of your own choosing. To optimize the appeal of the illusion, you may have to: (i) adjust the alignment of the images using photo editing software (or MATLAB) and (ii) adjust the lowpass and highpass cutoff frequencies by varying the standard deviations of the Gaussians used to generate the respective filters. Beyond technical correctness, marks will be awarded for the appeal of the illusion, so be creative.

4 Phase swapping (10 points)

1. **[5 points]** Select two images of the same dimensions, one containing a city scene and the other containing a human face. Compute their discrete Fourier transforms, swap their phase information and compute the inverse Fourier transforms. More specifically, if image1 has amplitude1 and phase1, and image2 has amplitude2 and phase2, then the final image1 will have amplitude1 and phase2 and the final image2 will have amplitude2 and phase1. Display the new output images using `imshow`.

2. **[5 points]** Describe the final images generated by swapping the phase information.

5 Canny edge detection (Total 30 for CPS843 and 50 for CP8307)

1. **[25 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 lecture notes. Your function should take as input a grayscale 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. **[20 points]** (Bonus for CPS843, not bonus for CP8307) 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.