Submission Requirements and Description (Very Important !)

Format requirements

(i) Please use the provided *Latex template* to write your report, and the report should contain your name, student ID, and e-mail address;

(ii) You should choose between Matlab and python to write your code, and provide a README file to describe how to execute the code;

(iii) Pack your **report**.pdf, **code** and **README** into a zip file, named with your student ID, like MG1833001.zip. If you have an improved version, add an extra '_' with a number, like MG1833001_1.zip. We will take the final submitted version as your results.

Submission Way

(i) Please submit your results to email cvcourse.nju@gmail.com , the email subject is "Assignment 1";

(ii) The deadline is 23:59 on March 26, 2019. No submission after this deadline is acceptable.

About Plagiarize

DO NOT PLAGIARIZE! We have no tolerance for plagiarizing and will penalize it with giving zero score. You may refer to some others' materials, please make citations such that one can tell which part is actually yours.

Evaluation Criterion

We mainly evaluate your submission according to your code and report. Efficient implementation, elegant code style, concise and logical report are all important factors towards a high score.

1 Linear Filters (30 points)

In class, we introduced 2D discrete space convolution. Consider an input image $I[i, j]$ and an $m \times n$ filter $F[i, j]$. The 2D convolution $I * F$ is defined as

$$(I * F)[i, j] = \sum_{k,l} I[i - k, j - l]F[k, l]. \tag{1.1}$$

Hint: The above operation is run for each pixel (i,j) of the result.

(a) Convolve the following I and F. Assume we use **zero-padding** where necessary.

$$I = \begin{bmatrix} 2 & 0 & 1 \\ 1 & -1 & 2 \end{bmatrix}, F = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \tag{1.2}$$

Important: DO NOT use Matlab for this question. Please show all necessary steps to receive full credit.

(b) Note that the F given in (1.2) is **separable**; that is, it can be written as a product of two 1D filters: $F = F_1F_2$. Here, we have

$$F_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, F_2 = \begin{bmatrix} 1 & -1 \end{bmatrix} \tag{1.3}$$

Compute $(I * F_1)$ and $(I * F_1) * F_2$, i.e. first perform 1D convolution on each column, followed by another 1D convolution on each now.
Important: DO NOT use Matlab for this question. Please show all necessary steps to receive full credit.

(c) Prove that for any separable filter $F = F_1F_2$,

$$I * F = (I * F_1) * F_2 \tag{1.4}$$

Hint: Expand equation (1.1) directly.

(d) In this question, we will count the exact number of multiplications (including those multiplications due to zero-padding) involved in part (a) and (b).

(i) When counting part (a), what's the number of multiplications if you flip and shift I?

(ii) Is the number of multiplications the same if you flip and shift F?

(iii) What are the number of multiplications for part (b)?

(iv) Which one of the two methods (a) or (b) requires fewer calculations?

Hint:

- The answers should be exact numerical values. We will not accept any approximation.

(e) Consider a more general case: I is an M_1 by N_1 image, and F is an M_2 by N_2 separable filter.

(i) How many multiplications do you need to do a direct 2D convolution?

(ii) How many multiplications you need to do 1D convolutions on rows and columns?
Hint: For (i) and (ii), the results should be two functions of M_1, N_1, M_2 and N_2. We will not accept any approximation.

(iii) Use Big-*O* notation to argue which one is more efficient in general: direct 2D convolution or two successive 1D convolutions?

2 Convolution (30 points)

Write a function that convolves an image with a given convolution filter

$$function[img1] = myImageFilter(img0, h) \tag{2.1}$$

As input, the function takes a greyscale image $(img0)$ and a convolution filter stored in matrix h. The output of the function should be an image $img1$ of the same size as $img0$ which results from convolving $img0$ with h. You can assume that the filter h is odd sized along both dimensions. You will need to handle boundary cases on the edges of the image. For example, when you place a convolution mask on the top left corner of the image, most of the filter mask will lie outside the image. One solution is to output a zero value at all these locations, the better thing to do is to pad the image such that pixels lying outside the image boundary have the same intensity value as the nearest pixel that lies inside the image.

You can call Matlab's function to pad array. However, your code can not call on Matlab's $imfilter$, $conv2$, $convn$, $filter2$ functions, or any other similar functions. You may compare your output to these functions for comparison and debugging. This function should be vectorized. Examples and meaning of vectorization can be found here. Specifically, try to reduce the number of for loops that you use in the function as much as possible.

3 Histogram equalization (40 points)

3.1 Global Histogram equalization (20 points)

Compute the gray level (luminance) histogram for an image and equalize it so that the tones look better (and the image is less sensitive to exposure settings). You may want to use the following steps:

(i) Convert the color image to luminance (Section 3.1.2 in textbook).

(ii) Compute the histogram, the cumulative distribution, and the compensation transfer function (Section 3.1.4).

(iii) (Optional) Try to increase the "punch" in the image by ensuring that a certain fraction of pixels (say, 5%) are mapped to pure black and white.

(iv) (Optional) Limit the local gain $f^{'}(I)$ in the transfer function. One way to do this is to limit $f(I) < \lambda I$ or $f^{'}(I) < \lambda$ while performing the accumulation (Equation 3.9), keeping any unaccumulated values "in reserve".

(v) Compensate the luminance channel through the lookup table and re-generate the color image using color ratios (equation 2.116).

(vi) (Optional) Color values that are clipped in the original image, i.e., have one or more saturated color channels, may appear unnatural when remapped to a non-clipped value. Extend your algorithm to handle this case in some useful way.

3.2 Local Histogram equalization (20 points)

Compute the gray level (luminance) histograms for each patch, but add to vertices based on distance (a spline).

(i) Build on Exercise 3.1 (luminance computation).

(ii) Distribute values (counts) to adjacent vertices (bilinear).

(iii) Convert to CDF (look-up functions).

(iv) (Optional) Use low-pass filtering of CDFs.

(v) Interpolate adjacent CDFs for final lookup.

Hint:

- Please refer to related section in textbook Computer Vision: Algorithms and Applications, 2010.
- Similar to Exercise 2, your code can not call on system histogram equalization function.
- For Exercise 2, 3.1 and 3.2, please test your code on Lenna image.