CE7491 Graded Lab 1

1. Quantization

Images `'Lena.bmp'`, `'Peppers.bmp'`, and `'Mandrill.bmp'` are all grayscale images with intensity ranging from [0-255]. Read in these images and quantize their *intensities* such that the output images contain 64, 16, 8, 4 and 2 *unique* levels, respectively.

Write your code so it loops through each on of the quantization level and shows these quantized images as figures. Your solution should pause (i.e. wait for a keypress) before it moves to the next quantization level. Matlab displays images as [0-255], so you will have to rescale your values to be between [0-255].

Some useful matlab commands: `pause, figure, imread, imshow`
Note, `imshow` can do the rescaling for you.

2. Resizing

Images `'Lena.bmp'`, `'Peppers.bmp'`, and `'Mandrill.bmp'` are all grayscale images sized 512x512. Read in these images and reduce their *sizes* by 2, 4, 8, and 16 times, respectively. Subsequently show these images at the *same display size*, by setting appropriate magnification factors. As with problem #1, have your matlab script show each images as a figure and then pause before it moves to the next quantization level.

Useful matlab command: `imresize`

3. Local Histogram Equalization

Matlab provides a function for image histogram equalization called `histeq`. In its basic form, histeq is used as follows:

```
J = histeq(I); % J is the result and I a grayscale image
```

Please read the documentation on this function carefully, its default setting is slightly different from what was discussed in class.

The global histogram obtained from all the pixels in the input image `I` is used to compute the equalization. Write a new function that performs "local histogram equalization". Instead of using the entire image's histogram, for each pixel in the image, your function should compute its new value based on the histogram computed over the pixels in a specified neighbourhood. The extent of this neighbourhood is defined by a window and should be a parameter to your function. (Note that larger window's give better results, why is that?).

Example function format:

```
J = localhisteq(I, [21 21]);  % where 21x21 is the local window size.
```

Your function should return an image `J` that has the same dimensions as `I`. When computing the local histogram of input pixels (x,y) where the windowed pixels are outside the input image, you should assume the pixels are reflected about the image borders. See figure 1 below for an example. Try your function on the images provided with varying window sizes (you may assume odd sized windows). Depending on your implementation, the local histogram equalization can be very slow, please have a few examples already prepared.

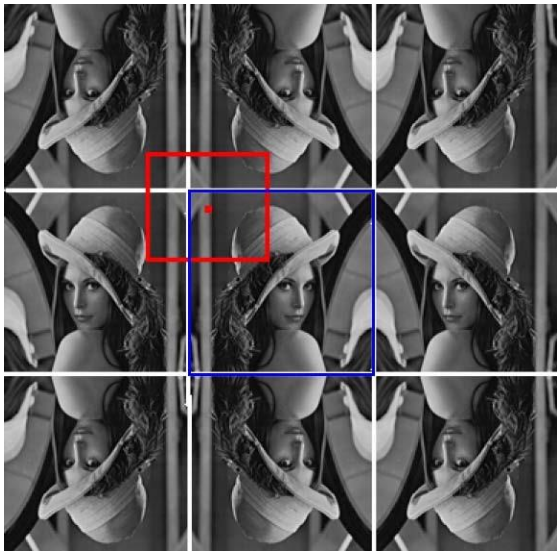


Figure 1. Reflected pixels.

4. Histogram Transfer

You have already learned about histogram equalization. In this problem, the task is not to flatten the histogram of an image, but rather manipulate the image to have a histogram that is identical (or as closely similar) to a reference histogram. For this problem, the reference histogram will be that of the second input image.

Your function should take two images, image1 and image2, and transfer the histogram of image 1 to image 2.

An example is seen here:

```
NewImage = histTransfer(I1, I2);
```

Image 1

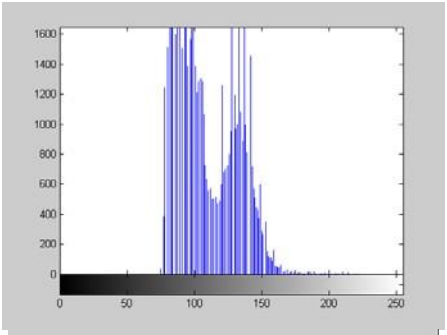
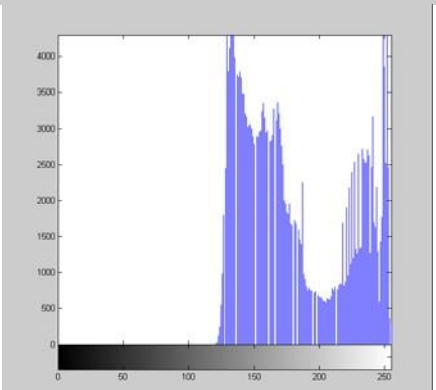
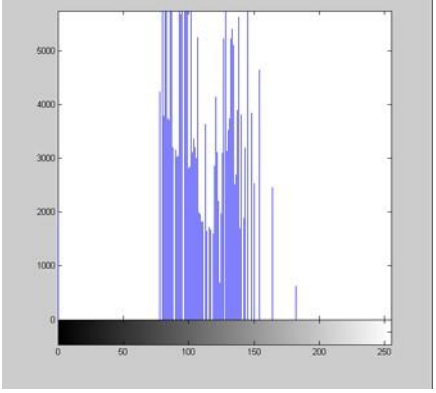


Image 2



"newImage"

Image 2 with Image 1's histogram



One way to verify if your algorithm is working correctly is to transfer the histogram of an image to itself. The resulting image should look similar to the original (although it might not be perfect).

Images "*pout.tif*", and "*bright_flower.tif*" are provided as examples. Your results may differ slightly from the one above, but the overall result should be roughly the same.