Computer Science 570/670 Computer Vision

Assignment 3 Due Tuesday, March 13

For this assignment, you should turn in all of the images you produce, and the matlab code for the varioius parts of the problem. Please include the images and the matlab code sequentially, and number them according to the problem numbers.

Part 1. Intro to brightness normalization. If an image contains more distinct brightness values than can be displayed on a particular display device, or an image contains more brightness values than can be seen by a human being, then it may be advantageous to adjust the image so that more of the information in the image can be seen. A simple example of this would be taking a checkboard made out of black squares and dark grey squares, and repainting it so that the dark grey squares are white.

There are two basic types of strategies for doing brightness normalization: changing the mapping of numbers to colors (changing the colormap in a colormap based system) or changing the values in the image and using the same colormap, as would be done in a "true color" mode (a mode that does not use any color map, but rather displays pixels directly as a function of the number contained in the pixel). In the first part of this assignment, you will explore brightness normalization techniques.

1. Download the file rawImage.dat from the class web page.
2. This file contains a header followed by 65536 pixels. Each pixel takes two bytes.
3. Use the command `fopen` to open the file and `fread` to read the bytes in the header. Throw the bytes in the header away.
4. Use `fread` again to read in the image pixels into an array.
5. Using the `reshape` command, reshape the pixels into a 256x256 matrix of pixels, and display the image using `imagesc`.
6. The image should be flipped on its side. Fix the image so that the round part of the figure (which is the top of the patient's head) is at the top of the image.
7. Use the command `hist` to look at the distribution of brightness values in the image. You may want to use the argument which increases the number of bins in the histogram.
8. Write a function which takes two inputs (an image, and a number of gray values) and returns another image such that there are about the same number of brightness values for each brightness. It may not be possible to have the exact same number of pixels at each brightness value, but the number of pixels for each brightness value should be the within 1 of each other. In some cases, you may need to map the same value in the original image to multiple values in the destination image. Use this function to render the given image as a 1-bit image, a 2-bit image, a 3-bit image, and so on, up to an 8-bit image.

9. Write another function which, given an image and a number of colors (as an integer), returns a grayscale "colormap" for displaying the image that makes the appearance of the image as much like the result of the previous part as possible. A "colormap" is a matrix with an arbitrary number of rows, but with 3 columns. Each row represents a "red-green-blue" color triple. Type "help colormap" to read about colormaps. Also, try typing `jet` or `gray` to look at two colormaps that are predefined in matlab. To use the colormap that you have created (suppose your colormap is names "foo"), use `imagesc` to plot your image, and then type `colormap foo`. The idea is, without changing the values of any of the pixels, to spread the brightnesses out as uniformly as possible.

Part 2. Local methods of brightness normalization. The human eye has the ability to separately control brightness normalization in different parts of an image. This is called "locally adaptive" brightness normalization. One way to do this is to try to boost the image contrast in areas where it is not very high.

1. Write a function which, given a "sub-image", that is a chunk from a larger image, returns an image whose smallest value is 0 and whose largest value is 256. The function should work by subtracting the minimum value in the sub-image from every pixel, and then scaling the image by a single constant factor so that the largest value is 256.

2. Use this function to write another function, localScale(), which takes 16x16 patches of an image, and replaces them with their locally normalized versions. Run the function on the original image for this problem. Notice the "patchyness" of the image.

3. Finally, write a "smoother" local gain correction algorithm which evaluates an offset (based on the minimum pixel in a patch) and a scaling factor (again based upon the range of the pixels in a patch) at EVERY pixel in the image. Then it uses this particular offset and scaling factor only at the pixel at the center of each patch. Rerender the image and notice its smoother appearance.