Rensselaer Polytechnic Institute
Department of Electrical, Computer, and Systems Engineering
ECSE 4540: Introduction to Image Processing, Spring 2018

Homework #7: due Monday, Apr. 2nd, at the beginning of class.
Show all work for full credit!

For this homework, you'll need the images in the file `http://www.ecse.rpi.edu/~rjradke/4540/hw7.zip`.

1. (20 points.) Consider the image `hotsauce.png` and the small template `letter.png`.

   (a) Use `normxcorr2` to match the template to the image and visualize the magnitude of the result as an image. It may help to use `colormap(jet)` to see the peaks. Interpret the regions where the magnitude of the cross-correlation is large.

   (b) Plot the peaks where the cross-correlation is greater than 0.9 as yellow rectangles the same size as the template on top of the original image. There are several ways to do this, but the functions `regionprops` and `rectangle` may be useful. Are all the peaks "correct" locations of the template?

   (c) Now try reducing the threshold to 0.8. Explain the phenomena you see in detail.

   (d) Now try reducing the threshold to 0.7. Explain the phenomena you see in detail.

2. (20 points.) In this problem you'll explore feature matching using your own images and the built-in functions in Matlab's Computer Vision System Toolbox.

   (a) First, choose a relatively cluttered environment with lots of corners and texture for an automatic feature detector to pick up on (e.g., your desk with a bunch of books and papers on it, or your living room or kitchen). Take one picture of this environment, take 2–3 steps parallel to the scene, then take another picture. Use the `detectHarrisFeatures`, `extractFeatures`, and `matchFeatures`, and `showMatchedFeatures` functions to display Harris feature matches between these two images (following the tutorial in the `matchFeatures` documentation). Critically assess the results; for example, what kinds of features are picked up? How well-distributed are the features across the scene?

   (b) Now, instead of using a second image that is quite similar to the first, move substantially around the scene (e.g., rotating your camera and getting closer to/further from the scene, so a square of pixels in one image will not easily match a square of pixels in the other image). Repeat your experiment from part (a). Do the results get worse?

   (c) Using the same images in part (b), use the `detectSURFFeatures` to detect and match SURF features using the same steps as in part (a). Critically assess the results; for example, what kinds of features are picked up? Are there a lot of good matches? Are there a lot of bad matches?

More fun on the next page ⟶

3. (20 points.) Write a Matlab function that takes an 8-bit image as input, and returns the following five values:

   - The entropy of the input image
   - The entropy obtained by coding horizontally adjacent (non-overlapping) pairs of pixels, divided by 2 since we have half as many double-size "pixels"
   - The entropy obtained by coding vertically adjacent (non-overlapping) pairs of pixels, divided by 2 since we have half as many double-size "pixels"
   - The entropy of the image obtained by taking the horizontal differences between adjacent pixels (wrapping from the end of one row to the beginning of the next)
   - The entropy of the image obtained by taking the vertical differences between adjacent pixels (wrapping from the end of one column to the beginning of the next)

   While there's a built-in Matlab function called `entropy`, you can't use it directly, since (for example) the difference images will no longer contain positive integers. I suggest you write an entropy function from scratch that operates on generic integer (e.g., `int32`) input. You can make new "images" with a larger dynamic range for the second two computations using something like `256*(left pixel) + (right pixel)`. The syntax `im(:)` may be useful here; i.e., rearranging a square matrix, column by column, into a single long column vector.

4. (20 points.) Apply your function above to the three input images in `noise.png`, `stripes.png`, and `cells.png`. Provide the five entropy values in each case, and (most importantly) interpret the results. Note that each of these images only has 4 gray levels, so without any coding, we would use 2 bits per pixel. Each entropy corresponds to the best-case number of bits per pixel we could hope to obtain using the corresponding image/scheme. Clearly, you need to look at the source images and think about the corresponding probability distributions in order to interpret the entropies.

5. (20 points.) For the `cells.png` example above, I computed the histograms of original values and vertical differences as follows:

Value	0	1	2	3
Count	67178	247264	180960	67098

Vertical difference	-2	-1	0	1	2	3
Count	54	8791	545770	7124	508	252

   Determine the Huffman codes and corresponding average bits per pixel corresponding to each distribution; how do these compare to the entropies you computed in the previous problem? Note that you can solve this problem before actually getting your Matlab code working.