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

Homework #8: due Monday, Apr. 16th, 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/hw8.zip`.

1. (25 points.) Write a function named `myjpeg` in Matlab that approximates the way the JPEG image compression algorithm works, using the following steps:

   - Input the source image and a positive number τ that determines the amount of compression (a bigger number means more compression).
   - Make sure the image is a double in the range [0, 255].
   - Subtract 128 from the image intensities (so they range between -128 and +127).
   - Split the source image into 8x8 blocks, and take the 2D Discrete Cosine Transform of each block. Use the functions `dct2` and `blockproc`.
   - For each block, set any coefficients that are less than τ in magnitude to 0.
   - Take the inverse DCT of each 8x8 block to recover a compressed image.
   - Add 128 to the resulting image intensities. Clip values to the [0, 255] range.
   - Display the compressed image on the screen.

   Turn in a code listing with your homework. (Note that your algorithm is doing threshold coding, while JPEG uses a kind of zonal coding based on a normalization matrix.)

   Test your routine on the image `guardian.png` provided in the zip file using $\tau = 10$, $\tau = 25$, and $\tau = 75$. Include a printout of each image with your homework. Explain what you see in each case (it will help to put the original image side-by-side with the compressed one; look carefully at fine-detail regions of the image). Experiment to see how large you can make τ before you see perceptible differences in the image. How does the saved filesize change with τ?

2. (25 points.)

   (a) Create a point spread function that simulates motion blur using the command
   `PSF = fspecial('motion',21,11);`. This creates a PSF that is roughly a line 21 pixels long oriented at an angle of 11 degrees. Apply the motion blur filter to the image `death.png` using the `imfilter` command. Remember to supply an appropriate 3rd argument so that array values outside the borders are assumed to be white. You can should see that the image is blurred to the point that text and fine details are not readable.

   (b) The image `museum-blur.png` was blurred with the same point spread function as above, and there was no additional noise added to it (although be aware that saving the image as an 8-bit image actually introduces a very small amount of quantization noise). Since we know the PSF exactly and there is no noise added to the image, apply the inverse filter using `deconvwnr` to undo the blur. Discuss the restored image in terms of sharpness, artifacts, readability of text, contrast, etc.

   (c) You should still observe some problems that came from the inverse filter due to the quantization noise. Instead, supply `deconvwnr` with a very small noise-to-signal ratio of 0.00001. How does this restored image look?

More fun on the next page ⟶

3. (25 points.)

   (a) Consider the image `clock2.png`. Compute its Radon transform using θ ranging from $0°$ to $179°$ and $\Delta\theta = 2°$. Display the result in grayscale with axes labels (hint: use `radon` with full input and output arguments, and clip the image intensities to [0, 255], don't scale them). Interpret what you see, paying particular attention to the $\rho = 0$ line. How are the θ value and bright spots in the Radom transform related to features of the original image? Why is the image black at the top and bottom, and why is the non-zero region of the Radon transform rectangular?

   (b) Now recover an approximation to the original image using the inverse Radon transform. Remember that you must supply the `theta` vector to `iradon`. Interpret what you see. Where does the result look good and where are there visible artifacts? Repeat the experiment by taking the forward and inverse Radon transform using $\Delta\theta = 1°$. How has the result improved? Are there still visible artifacts?

4. (25 points.) Consider the grayscale image `guys.png`. In this problem, you'll look at four ways of rendering it in black and white. For each method, include the black-and-white image produced by your code, and discuss its qualitative appearance in some detail. Make sure to cast the input image as a `double` before starting!

   (a) First, threshold the image at a value of 128.

   (b) Next, apply uniform dithering by adding i.i.d. uniform random noise in the range [-50,50], then thresholding the resulting image at a value of 128.

   (c) Third, halftone the image by tiling the image into 4×4 blocks, adding the Bayer mask

$$16 \times \begin{bmatrix} 1 & 9 & 3 & 11 \\ 13 & 5 & 15 & 7 \\ 4 & 12 & 2 & 10 \\ 16 & 8 & 14 & 6 \end{bmatrix}$$

   to each block, and thresholding the resulting image at a value of 256.

   (d) Finally, use Matlab's built-in `dither` command to apply Floyd-Steinberg dithering.