EE 133 - Digital Image Processing
Department of Electrical and Computer Engineering
Tufts University Spring 2017
Problem Set #4

Distributed: March 7, 2017 Due: March 17, 2017

Problem 4.1

Compute the 2D Fourier transform of the continuous 2D signal

$$f(x,y) = \begin{cases} 1 & x > y \\ 0 & \text{else} \end{cases}$$

Problem 4.2

In this problem consider a 64×64 image defined as

$$g(m,n) = f(m,n) - \frac{1}{64^2} \sum_{m=1}^{64} \sum_{n=1}^{64} f(m,n)$$

where

$$f(m,n) = \begin{cases} 1 & m \geq n \text{ with } m,n = 1,2,\ldots 64 \\ 0 & \text{else} \end{cases} \tag{4.2.1}$$

1. What is the relationship between the 2D DFT of g and that of f?
2. Compute and plot the 2D DFT of g.
3. According to the results of the first problem, we would expect to see one line when we plot the DFT g, but we see three lines. First, for the expected line, what is the slope and why? Second, why the other two lines?

Problem 4.3

A very commonly used transform in image processing closely related to the Fourier is known as the *Discrete Cosine Transform* (DCT) of which there are four varieties known as DCT-I through DCT-IV. We will deal here with DCT-II for which the forward transform of an $N \times N$ image $f(m,n)$ into the transform domain image $F(u,v)$ is

$$F(u,v) = \frac{2c(u)c(v)}{N} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \cos\left(\frac{2m+1}{2N}\pi u\right) \cos\left(\frac{2n+1}{2N}\pi v\right) f(m,n) \tag{4.3.2}$$

with $c(k) = 1/\sqrt{2}$ if $k = 0$ and $c(k) = 1$ for $k \neq 0$. In this equation, m, n, u and v are all integers from 0 to $N-1$. It turns out that the DCT is closely related to the DFT and, perhaps not too surprisingly, there are methods based on the Fast Fourier Transform to obtain Fast DCTs. In Matlab the function call is `dct2` with something similar is available via SciPY in Python. For the remainder of this problem, you can feel free to use these tools. That is, you *do not* have to write your own DCT routines (though you certainly may if you wish).

1. The DCT may be thought of as the representation of an $N \times N$ image $F(u,v)$ in terms of N^2 "basis images" $B_{m,n}(u,v)$. Identify these basis functions and, for an 8×8 image, plot them.

2. Show empirically[1] for $N = 8$ that these basis images are orthogonal meaning that

$$\sum_{u=0}^{N-1} \sum_{v=0}^{N-1} B_{m,n}(u,v) B_{p,q}(u,v) = \delta(m-p)\delta(n-q). \tag{4.3.3}$$

3. It turns out that (4.3.3) is true for any N though proving it rigorously is really quite tedious. Nonetheless, assuming that (4.3.3) is true, explain how to invert the DCT-II. More specifically provide an equation like (4.3.2) for finding $f(m,n)$ from $F(u,v)$. In Matlab, the relevant function is `idct2`.

4. To a (very) rough first approximation, the JPEG compression scheme operates according to the following algorithm

   (a) Break the image up into 8×8 blocks (for here we assume all images contain a multiple of eight rows and columns).

   (b) For each block, take the DCT-II and keep only those coefficients that are above some threshold setting all others to zero.

   (c) Encode the indices of the nonzero elements of each block along with the values and store this information in a file.

   To decode an image then, we merely reconstruct each block from the information in the file and take the inverse DCT to get an approximation of the original 8×8 block of pixels.

   For this problem, implement your own version of JPEG by breaking the image into blocks, taking the DCT, zeroing small elements and then taking the inverse DCT to recover the image. Test your method on the `cameraman` image included with this problem set being sure to explain in as quantitative a manner as possible how accuracy of the reconstructed image is impacted by the number of zero coefficients in the compressed image. The evaluation should be done using a wide range of thresholds and, with $f(m,n)$ denoting the exact image and $f_\tau(m,n)$ the one obtained after compression using a threshold τ, use the following metric as a measure of accuracy:

$$d(\tau) = \frac{\sum_{m,n}(f_\tau(m,n) - f(m,n))^2}{\sum_{m,n} f(m,n)^2}$$

Problem 4.4

Here we want to investigate how one can build gradient-based filters that are sensitive to edges in directions other than vertical and horizontal. As a first approach we can think about a brute force use of Fourier properties. Specifically, we know that in continuous space at least, the Fourier transform of a rotated version of an image is the rotation of the Fourier original transform. Mathematically:

$$\mathcal{F}\{f(Rr)\} = F(Rk) \tag{4.4.4}$$

[1] In other words, show this using Matlab as opposed to proving it with pencil and pape

where

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad r = \begin{bmatrix} x \\ y \end{bmatrix} \quad k = \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.4.5}$$

and θ is the angle of rotation.

1. Explain how this fact can be used to build filters are sensitive to edges oriented in a direction other than vertical or horizontal.

2. Suppose we used as our initial edge filter the derivative of a Gaussian:

$$h(x,y) = \frac{\partial}{\partial x}\left\{ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \right\} \tag{4.4.6}$$

   Find an expression for the edge filter that is oriented θ radians off of the x axis. Plot using Matlab both the filter itself as well as its Fourier transform (or the amplitude of the transform) to verify that your results are correct.

Problem 4.5

Here we look at a classic approach to building filters that are sensitive to edges in different orientations, known as "steerable filters." Zipped with this problem set is an old, but classic, paper on this topic. While you are certainly free to read the whole paper, this problem deals with bits and pieces that I hope are accessible to everyone in the class.

1. Equation (4) on page 3 of the paper says that you can build a steerable filter using linear combinations of x and y derivatives of a Gaussian. In the last problem, you showed a different way to build an oriented filter using just the x derivative. What is going on here? How are the two methods related?

2. Explain in words what the authors are trying to accomplish in Section 5.1 and what equation (19) has to do with this?

3. Using the G_4 and H_4 filters discussed in Appendix G, implement in Matlab the ideas in Section 5.1.1 and show that it works for the two test images in Figure 9 of the paper. Specifically for each of the images to be analyzed, you should:

   (a) Build the quadrature filters G_4 and H_4. You will have to figure out what the right level of discretization is here.

   (b) For all angles of interest, convolve these filters with the image under consideration to obtain G_4^θ and H_4^θ.

   (c) Compute $E_4(\theta)$ using the formula (18) in the paper. Note that for each θ, E_4 will be an entire image. Thus you will have a stack of images for this problem.

   Explain (including convincing plots) how the data in the E_4 image stack can be used to determine the orientation of one of the pixels in the vertical line as well as the distribution of orientations at the center pixel in the cross image. Try it now on a test image like that in Figure 9-b, but where the top half of the vertical line is removed (i.e. a "T"). Why are the results not correct?