ECE 203
Lab 8: Image Processing and Compression

Verification: Complete the warm-up section of the lab during your assigned lab time. Make sure an instructor signs the verification sheet found at the end of the lab report.

Lab Report: Submit a lab report for the entire lab using MATLAB publisher.

---

1 Introduction

The objective of this lab is to use computers to analyze, process and compress images. The two-dimensional Discrete Fourier Transform (DFT) is the main tool we will be using. The 2-d DFT can be rapidly computed using an algorithm known as the Fast Fourier Transform (FFT). MATLAB has a built-in function, `fft2`, that implements this algorithm. We will also look at a related transform called the Discrete Cosine Transform (DCT). The DCT is essentially the same idea (representing images in terms of two-dimensional waveforms), but instead of complex exponentials/sinusoids the DCT represents images in terms of purely real-valued cosines. This makes the DCT especially attractive for image compression (the DCT is the basis of the JPEG compression standard).

2 Overview

The 2-d DFT is the Fourier series representation for digital images. Let $x[m,n]$, $m = 0,\ldots,M-1$, $n = 0,\ldots,N-1$, denote an image of $M \times N$ pixels. The 2-d DFT coefficients of $x[m,n]$ are given by

$$X[k,\ell] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x[m,n]\, e^{-j2\pi\frac{k}{M}m}\, e^{-j2\pi\frac{\ell}{N}n}, \quad k = 0,\ldots,M-1,\ \ell = 0,\ldots,N-1.$$

The image $x[m,n]$ can then be represented or synthesized by the following weighted sum of complex exponential functions

$$x[m,n] = \frac{1}{MN}\sum_{k=0}^{M-1}\sum_{\ell=0}^{N-1} X[k,\ell]\ e^{j2\pi\frac{k}{M}m}\, e^{j2\pi\frac{\ell}{N}n}$$

The DFT coefficient $X[k,\ell]$ appropriately weights the corresponding complex exponential function, $e^{j2\pi\frac{k}{M}m}\, e^{j2\pi\frac{\ell}{N}n}$, a complex-valued two-dimensional sinusoidal waveform. There are two digital frequencies associated with this wave; $\widehat{f}_v = k/M$ cycles/(vertical sample) and $\widehat{f}_h = \ell/N$ cycles/(horizontal sample). To relate the digital frequencies to the corresponding

spatial frequencies simply multiply the digital frequencies by the number of pixels per meter (or whatever measure of spatial distance you like to use).

The DCT is very similar to the DFT except that it employs 2-d cosine waveforms, instead of complex exponentials. MATLAB has built-in functions `dct2` and `idct2` to compute the 2d DCT of an image. The 2d DCT coefficients are defined as follows for $k = 0, \ldots, M-1$ and $\ell = 0, \ldots, N-1$:

$$X_{dct}[k,\ell] \;=\; \beta_k \beta_\ell \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] \cos\left(\frac{\pi(2m+1)k}{2M}\right) \cos\left(\frac{\pi(2n+1)\ell}{2N}\right)$$

where β_k and β_l are 1 if k or ℓ, respectively, are equal to 0, and $\sqrt{2}$ otherwise. To recover $x[m,n]$ from $a_{k,\ell}$ we have a formula analogous to the DFT reconstruction:

$$x[m,n] \;=\; \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{\ell=0}^{N-1} \beta_k \beta_\ell \, X_{dct}[k,\ell] \cos\left(\frac{\pi(2m+1)k}{2M}\right) \cos\left(\frac{\pi(2n+1)\ell}{2N}\right)$$

3 Warm-up

3.1 Image Analysis using the 2-d DFT

Load the test image `camerman.mat`. Display the image and its 2-d DFT using the following commands:

```
load cameraman.mat
figure(1)
imagesc(x)
colormap(gray)
axis('square')
figure(2)
spec_x = fft2(x);
imagesc(fftshift(log10(abs(spec_x))));
colormap(gray)
axis('square')
```

You should observe prominate streaks or lines in the display of the image spectrum. Comment on what structures in the original image might be generating these spectral features.

Instructor Verification

3.2 Image Filtering

Edges can be detected by filtering the image with an appropriate filter. Consider the following filters defined by their point spread responses:

$$h_h = \begin{bmatrix} 1/4 & 1/4 \\ -1/4 & -1/4 \end{bmatrix}$$

$$h_v = \begin{bmatrix} 1/4 & -1/4 \\ 1/4 & -1/4 \end{bmatrix}$$

Apply these filters to the image using the command `y = conv2(x,h,'same')`, where h is either h_h or h_v. Display the resulting filtered images using the command `imagesc(abs(y))`; the absolute value will reveal locations where the filters had large positive or negative responses. Comment on the resulting images and the actions of the filters.

Alternatively, the edges can be obtained by lowpass filtering the image, and then computing the difference between the orginal and lowpass images. You can use the lowpass filter

$$h_{lp} = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}$$

Display the magnitude of the difference between the original and blurred image. Explain why this process reveals the edges in the image.

Instructor Verification

3.3 Image Filtering using the DFT

Convolution between the image and the filter point spread function is equivalent to multiplication in the frequency domain. Use the commands `fft2` and `ifft2` to compute the lowpass filtered image described above. To do this you will need to compute the 2d DFT of the filter, of a size compatible with that of the image. The image is 256×256 and `fft2(x)` will generate its 256×256 DFT. The filter's point spread response is only 2×2. To compute its 2d DFT at 256×256 points in frequency use the command `fft2(h_lp,256,256)`. Then you can multiply the two DFTs together and take the inverse DFT of the result. Again, display the magnitude of the difference between the original and blurred image to verify that your fft-based procedure is working.

Instructor Verification

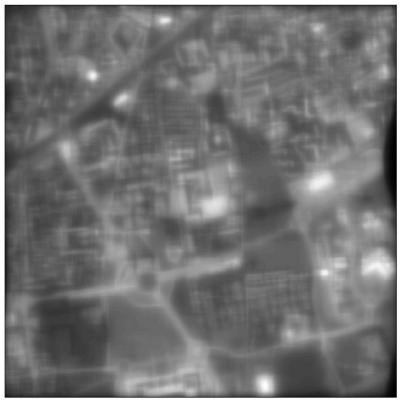
4 Image Processing, the DFT and the DCT

4.1 Image Restoration

This problem investigates image deblurring. An ideal sattelite image is depicted below (This is an image of Nimes, France). A blurred version is shown next to it. The blurring could be

caused by a number of factors including atmospheric distortions. Typically the blur is not this significant, but in adverse conditions it can be quite severe. Both images can be downloaded from course website. The file containing the ideal image is called `nimes_france.mat`.

ideal image



blurred image

The point spread function (PSF) $h[m, n]$ of the blurring operator is known in this case, and is contained in the file `blur.mat`. The inverse PSF can be derived from $h[m, n]$. The inverse PSF is in the file `blurinv.mat`. Both PSF files can also be downloaded from the course website.

a. Generate the blurred image by convolving the ideal image with the blurring PSF. Display the blurred image in Matlab.

b. Using the Matlab `conv2` function, apply the inverse PSF to the blurred image. Use the Matlab commands `tic` and `toc` to determine the run-time of this convolution operation.

c. Now use the fft to perform the deblurring. You will need to divide the 2d DFT of the blurred image by the 2d DFT of the blurring PSF and then take the inverse 2d DFT of the result (the division should be taken element-wise). The image size is 512×512, so you will need to generate the 2d DFT of the blurring PSF at this resolution (see warm-up exercise). Display the resulting restored image. Notice the small distortion effect near the boundaries of the image. This occurs because the fft does not yield exactly the same result as normal convolution. Compare the run-time in this case with the speed of `conv2` determined above.

d. Notice that the bottom and right edges of the image restored using the fft look strange. Let `xrecovery` denote the image restoration from (c) above. Modify this image using the following operation:

```
shift = exp(-j*2*pi*21/N*(0:N-1)')*exp(-j*2*pi*21/N*(0:N-1));
xrecovery1 = real(ifft2(fft2(xrecovery).*shift));
```

Compare `xrecovery1` with `xrecovery`. The new result should look better. Explain what the operation above is doing to the image and try to determine why this might be necessary.

e. Now suppose that a tiny bit of noise is added to the blurred image. Generate a slightly noisy and blurred image using the command

```
y = conv2(x,h,'same') + randn(size(x))
```

Small amounts of noise could be due to instrumentation noise, atmospheric effects, and quantization. The noise is imperceptible to the eye (compare the blurred image with and without noise), but has a dramatic effect on the deblurring process. Apply your fft-based deblurring process to this image. Comment on the result. Can you come up with an improved deblurring process that deals the noise more effectively? HINT: Let `Hf` denote the 512×512 DFT of the blurring PSF. Note that

```
fft(y)./Hf = fft2(y).*conj(Hf)./abs(Hf).^2
```

where `conj` computes the complex conjugate. Instead of the previous filter, consider the modified filter

```
fft2(y).*conj(Hf)./(abs(Hf).^2 + 0.001)
```

Vary the small constant in the denominator to improve the result. Explain why this can help to improve the restoration.

4.2 Image Quantization and Digital Watermarking

The data file `cam_wm.mat` contains an image called `y`. The image should look identical to the cameraman image, but it is slightly different than the image we used in warm-up. There is a secret digital 'watermark' in this image. The image pixels are represented by numbers between 0 and 255 (i.e., 8 bits of precision). The watermark is the pattern determined by the least significant bit (LSB) of each pixel.

a. Determine method for extracting the LSB of each pixel, and display the LSBs as an image. Assume that you only have access to the image `y`. Do not use the original camerman image in order to extract the watermark. You will then see the secret watermark (a binary image), which is otherwise invisible.

b. Using a digital image of your choosing, embed a watermark in the LSB and demonstrate that it can be recovered from the watermarked image.

4.3 Image Compression

Everyday you are using JPEG image compression (unless you happen to stay off-line). The basic principles behind JPEG are very easy to understand. The 2d DCT is the central processing step in the JPEG compression process. In a nutshell, the image to be compressed is sub-divided into 8×8 subimages, the DCT of each 8×8 block of pixels is computed, and the resulting DCT coefficients are quantized, coded and stored as a bit file. To decompress a JPEG file the process is inverted. The bits are converted to quantized DCT coefficients and the inverse DCT reproduces an approximation to the original 8×8 block (approximately because the quantization step is not perfectly invertible). This problem explores the basic concepts behind the JPEG standard.

a. Use the following code to display the 2d cosine waveforms associated with the 8×8 DCT. Explain how this code produces the cosine waveforms associated with the DCT. Which waveforms have the high-frequency? Does it make sense that any 8 image of pixels could be generated as a weighted combination of these waveforms?

```
N=8;
for m=1:N
    for n=1:N
        e=zeros(N,N);
        e(m,n)=1;
        b = idct2(e);
        subplot(8,8,m+(n-1)*N)
        imagesc(b)
        colormap(gray)
        set(gca,'Xtick',[])
        set(gca,'Ytick',[])
        axis('square')
    end
end
```

b. Load the image `cameraman.mat`. Compute the DCTs of every 8×8 image block using the code below. Display and explain the resulting image `y`.

```
[M,N] = size(x);
Mblocks = M/8;
Nblocks = N/8;
% compute DCT of 8x8 subimages
for m = 1:Mblocks
    for n=1:Nblocks
        Mrange = (m-1)*8+1:(m-1)*8+8;
        Nrange = (n-1)*8+1:(n-1)*8+8;
```

```
        block = x(Mrange,Nrange);
        DCTblock = dct2(block);
        y(Mrange,Nrange) = DCTblock;
    end
end
```

c. The JPEG compression standard is based on encoding (with bits) the array `y` computed in part (b) above. To do this, the values in `y` are quantized to integer values. This is accomplished by applying the following operation to each DCT block:

```
quantized_DCTblock = round(DCTblock./Q);
```

where `Q` is a 'quantization matrix' of scaling factors that reduces the values of high-frequency DCT coefficients relative to low-frequency coefficients. This is done because the human eye is less sensitive to high frequency brightness variation. Here is a typical quantization matrix:

```
Q = [16 11 10 16 24 40 51 61;
     12 12 14 19 26 58 60 55;
     14 13 16 24 40 57 69 56;
     14 17 22 29 51 87 80 62;
     18 22 37 56 68 109 103 77;
     24 35 55 64 81 104 113 92;
     49 64 78 87 103 121 120 101;
     72 92 95 98 112 100 103 99];
```

The result is that high-frequency coefficients are often quantized to values near 0, and this means that fewer bits are needed to store the quantized DCT coefficients than the original image pixels. Each pixel in the original image ranges between 0 and 255 in value, and therefore requires 8 bits. After the quantization process, most of the DCT coefficients will be zero (requiring essentially 0 bits) and the non-zero DCT coefficients will require 8 bits (just like the pixels). JPEG does something a bit more complicated with the quantized DCT coefficients, but exploits the same basic idea.

The compression ratio of the JPEG encoded image is approximately equal to the number of non-zero quantized DCT coefficients divided by the total number of quantized DCT coefficients (the total is equal to the total number of pixels in the original image). Therefore the compression ratio is

$$\text{compression ratio} \;=\; \frac{\#\text{ non-zero quantized DCT coefficients}}{MN}$$

Display the quantized DCT coefficients and compare them to the unquantized coefficients. Compute the compression ratio.

d. Decompression refers to the process of reconstructing the image from the quantized DCT coefficients. Devise a decompression process. Hint: You will need to approximately undo the effect of quantization and then compute the inverse DCT of each block using `idct2`. Note that the quantization cannot be exactly inverted, especially if a DCT coefficient was quantized to 0. This causes errors in the reconstruction of the image, and for this reason JPEG compression is called *lossy* (because it doesn't perfectly preserve all the information in the image).

The compression ratio can be increased/decreased by multiplying the quantization matrix Q by a scalar factor greater/lesser than 1. Vary this factor and compare the results in terms of the compression ratio and visual quality of the decompressed image. Discuss the nature of artifacts/distortions caused by the JPEG compression process.

Instructor Verification – Lab 8

Name: _____

Verification 1: _____ Date: _____

Verification 2: _____ Date: _____

Verification 3: _____ Date: _____