EE113: Course Project Spring 2016

Prof. Mihaela van der Schaar **Due date: May 23, 2016**

DSP in action: Image processing using MATLAB

Project Summary

Have you ever wondered how *Photoshop* applies effects on images? How *Facebook* recognizes faces in photos? They simply use DSP techniques! In this project, we will try to understand how can we use the concepts that we have learned in the course to carry out various practical image processing tasks. The primary objectives of this project are:

- Sharpen your knowledge of DSP techniques by utilizing them in applications based on basic image filtering principles.
- Learn basic syntax and functionality of MATLAB. You may benefit from acquiring this skill in other courses as well!

Project Instructions

- The project is worth **14 points**.
- You can work on the project individually or in groups of **2 students**.
- Each group will submit one project report on **May 23, 2016**. **No late submissions are allowed!**
- All required image files, in addition to a brief MATLAB tutorial, can be found in the project file **"Project.rar"** on CCLE.
- Please submit a soft copy of your report in pdf format through CCLE. No hard copy needed! Submit your report via the "Project" link in the EE113 homepage on CCLE. **The submission link will expire once the deadline is overdue!**

MATLAB functions

Throughout the project, you will be instructed to use specific built-in MATLAB functions in order to implement the requirements of the project. Type **help "function"** in the MATLAB command prompt for information on how to use a specific function. For example, typing **"help mean"** into the command prompt will provide you information about how to use the function **"mean"**. You may also search for information about specific functions on the Internet.

1 Introduction

In this course, you have studied one-dimensional discrete-time sequences $x(n)$, which are indexed by the discrete-time variable n. In this project, instead of working with one-dimensional discrete-time sequences, you will work with images. Images can be represented as two-dimensional $M \times N$ sequences $f(x, y)$, where x $(0 \leq x \leq M-1)$ and y $(0 \leq y \leq N-1)$ are spatial coordinates, which denote the row and column indices, respectively, of the 2-D image array $f(x, y)$. Many of the computations you are used to performing on 1-D sequences, such as convolution and Fourier transform, can be extended to 2-D images.

Two-dimensional discrete Fourier transform (2-D DFT)

The one-dimensional DFT describes the frequency content of a one-dimensional signal $x(n)$. Similarly, the two-dimensional discrete Fourier transform describes the frequency content of a two-dimensional signal (or image) $f(x, y)$. An image with smooth color gradients (e.g. an image of a sky at sunset) will have almost all of its energy concentrated in the lower frequency components of the image; in contrast, an image with fine-grained textures (e.g. an image of a forest) or with many edges (e.g. an image of a cathedral) will have significant energy concentrated in the higher frequency components.

Extension of the one-dimensional DFT and its inverse to two dimensions is straightforward. The discrete Fourier transform of an image $f(x, y)$ of size $M \times N$ is given by the equation

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}. \tag{1}$$

As in the 1-D case, this expression must be computed for $u = 0, 1, \ldots, M-1$ and $v = 0, 1, \ldots, N-1$. Similarly, given $F(u, v)$, we obtain $f(x, y)$ via the inverse Fourier transform, given by the expression

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}, \tag{2}$$

for $x = 0, 1, \ldots, M-1$ and $y = 0, 1, \ldots, N-1$. The above two equations comprise the two-dimensional, discrete Fourier transform (2-D DFT) pair. The variables u and v are the transform or frequency variables, whereas x and y are the spatial or image variables. The frequency variables u and v denote the vertical and horizontal frequencies, respectively.

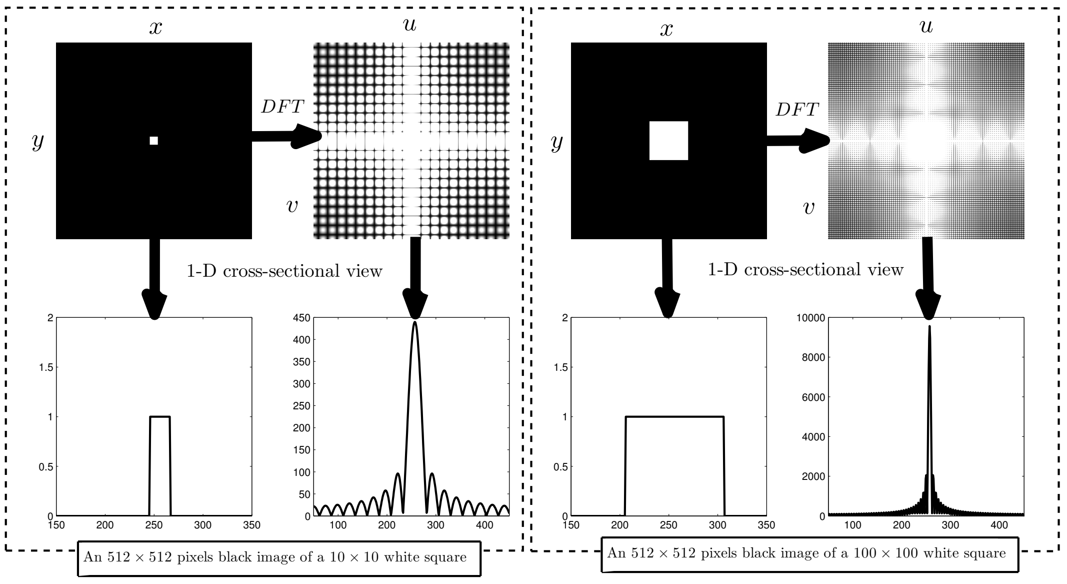

Figure 1: Depiction for DFT transforms for images that resemble square wave signals.

Figure 1 shows a 512 × 512-pixel black image with a white square in the center. One can think of this image as a 2-D analog of the conventional 1-D square wave signal, whose Fourier transform is a Sinc signal. As we can see in Figure 1, applying 2-D DFT of the square image leads to another image that resembles a 2-D Sinc function that has a main lobe at the center of the image, surrounded by side lobes. As it is for classical 1-D signals, changing the width of the square signal (i.e. the area of the width square) affects the width of the main lobe of the Sinc function in the frequency domain. By looking at a cross-sectional view of any of the images, i.e. a row of pixels, we can see a classical 1-D square signal and a 1-D Sinc signal. Similarly, in Figure 2 we can see that for a black image with two white pixels around the center, which resembles two unit samples, applying inverse DFT, we retrieve an image that is similar to a sinusoid, i.e. a wave of alternating black and white pixels with a frequency that depends on the spacing between the two unit samples. This is analogous to the frequency domain representation of the sinusoid, which is simply two unit samples centered around zero.

Two-dimensional Convolution

As in the 1-D case, the foundation of linear filtering in both the spatial and frequency domains is the convolution theorem, which may be written as

$$f(x,y) \star h(x,y) \overset{\text{DFT}}{\Leftrightarrow} F(u,v)\,H(u,v), \tag{3}$$

and, conversely,

$$f(x,y)\,h(x,y) \overset{\text{DFT}}{\Leftrightarrow} F(u,v) \star H(u,v), \tag{4}$$

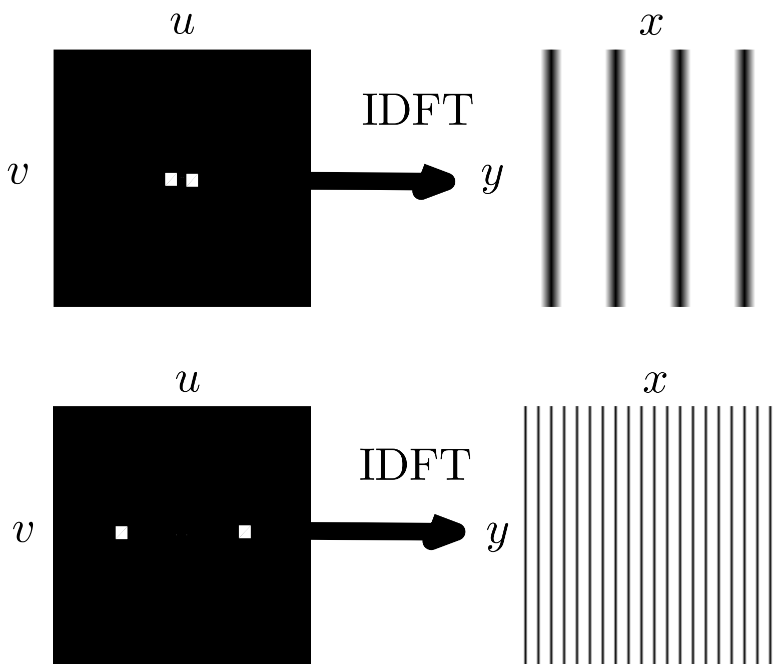

Figure 2: Depiction for IDFT transforms of unit sample functions resulting 2-D sinusoidal images in the frequency domain.

where

$$f(x,y) \star h(x,y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n)\, h(x-m, y-n) \tag{5}$$

is the convolution of $f(x,y)$ and $h(x,y)$.

2 Project Requirements

Part I: Image Blurring using Moving Average Filters (3.5 points)

Most photo editing programs allow you to apply a blurring effect to an image. How is that effect implemented? In this exercise, we will use filtering and convolution concepts to implement image blurring!

Complete the following tasks and include the requested information in your report.

Step 1. Load and display images in MATLAB: Make sure that the current directory for your Matlab workspace contains the image files "Lena.jpg" and "cameraman.tif" (you will find these images in the "Project.rar" folder on CCLE). Create a new MATLAB m-file in order to write your code. Use the MATLAB function "imread" to store the pixel values for the two images "Lena.jpg" and "cameraman.tif" in two matrices. In your report, include the following:

1. Display the "Lena.jpg" and "cameraman.tif" images using the "imshow" function. **(0.25 point)**
2. Determine the width and height of the "Lena.jpg" and "cameraman.tif" images in pixels (use the "size" command). **(0.25 point)**
3. Display the frequency transform of the "Lena.jpg" image using the command

   $$\log(1 + \text{abs}(\text{fftshift}(\text{fft2}(A)))).$$

   This is the logarithm of the Fourier transform for an image stored in matrix A. We are taking the absoulte value of the Fourier transform since it is complex-valued, and we are then taking the logarithm to enhance the visibility of the spectrum. (Search for an explanation for the operation of both "fft2" and "fftshift" using MATLAB help. The frequency transform is itself an image that you should display using "imshow"). **(0.5 point)**

Step 2. Image Blurring: In this exercise, we will apply an image blurring effect to both the "Lena.jpg" and "cameraman.tif" images using moving average filters. We consider the following filters:

$$h_1 = \frac{1}{100}\begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}_{10\times 10}, \text{ and } h_2 = \frac{1}{2500}\begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}_{50\times 50}$$

where h_1 is a 10×10 matrix, and h_2 is a 50×50 matrix. All the entries of h_1 are equal to $\frac{1}{100}$, and all the entries of h_2 are equal to $\frac{1}{2500}$. In your report, include the following:

1. Create matrices h_1 and h_2 using the "ones" function in MATLAB. Use the "conv2" function to apply convolution of the images "cameraman.tif" and "Lena.jpg" with each of the filters h_1 and h_2. Using the "imshow" command, display the images you get after convolving the cameraman and Lena images once with h_1 and once with h_2. Comment on the images you get. **(2 points)**

   (Hint: you need to convert the images into double format and store them in a matrix before using the command "conv2". You can do that using the following command: "A = im2double(imread('cameraman.tif'))".)

2. Compare the size of the "cameraman.tif" image with the size of the image that results from the convolution of the "cameraman.tif" image with the filter h_2. Which one is larger and why? **(0.5 points)**

Part II: Edge Detection using First Difference Filters (3.5 points)

In Part I, we have applied the filters h_1 and h_2 to the "cameraman.tif" image, and you should have observed that those filters apply "smoothening" to the images, i.e. they create the image blurring effect that you can apply using any photo editing application. In many other applications, including object recognition, one is interested in finding the edges and boundaries in images. In this exercise, we will detect edges in an image using *difference filters.*

We consider the following "first difference" filter

$$h_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

In your report, include the following:

1. Using the "conv2" function, apply the convolution operation between each of the images "cameraman.tif" and "Lena.jpg" with the filter h_3 and display the two output images. Comment on the images you obtain, and compare them with the images that you obtained in Step 2, Part I. **(1 points)**

2. Based on your understanding of the convolution operation, explain the difference between the filter h_1 and the filter h_3, i.e. what are the different effects they introduce in the input image? which one of them is a low pass filter and which is a high pass filter? Support your arguments by displaying the frequency transforms of h_1 and h_3 using "fft2" and "fftshift". **(1 points)**

3. Now take the image that results from convolving "cameraman.tif" with the filter h_3. This is a distorted version of "cameraman.tif". We want to recover the original "cameraman.tif" image from this distorted image. Use the MATLAB commands "fft2" and "ifft2" to recover "cameraman.tif" from the distorted cameraman image that you obtained in (1). **(1.5 points)**

Part III: Image Resizing through Up-sampling and Interpolation (3.5 points)

Computers allow you to zoom in an image or expand its size. We always notice that image zooming or expansion is usually accompanied with "pixelization", i.e. the image becomes of a lower resolution. Why is that happening? In this exercise, we explore techniques for resizing (zooming) an image while maintaining good resolution. These techniques will be based on our knowledge of up-sampling and interpolation filtering.

Before discussing image resizing, we briefly wrap up the basic concepts of up-sampling and interpolation. Recall that for a sequence $x(n)$, an up-sampled version of that sequence with factor L is denoted by $x_L(n)$ and is constructed by inserting $L-1$ zeros between the samples of $x(n)$. Figure 3 depicts a sequence $x(n)$ (plotted in the left) and up-sampled version of that sequence with a factor of 5 ($x_5(n)$ is plotted in the middle). The sequence $x_5(n)$ is an expanded version of $x(n)$, where the extra samples in $x_5(n)$ are padded with zeros. Interpolation is a filtering technique that fills in those extra samples with non-zero values. The simplest interpolation filter is the zero-order-hold filter, which just repeats the samples of $x(n)$ 5 times as shown in Figure 3 (right).

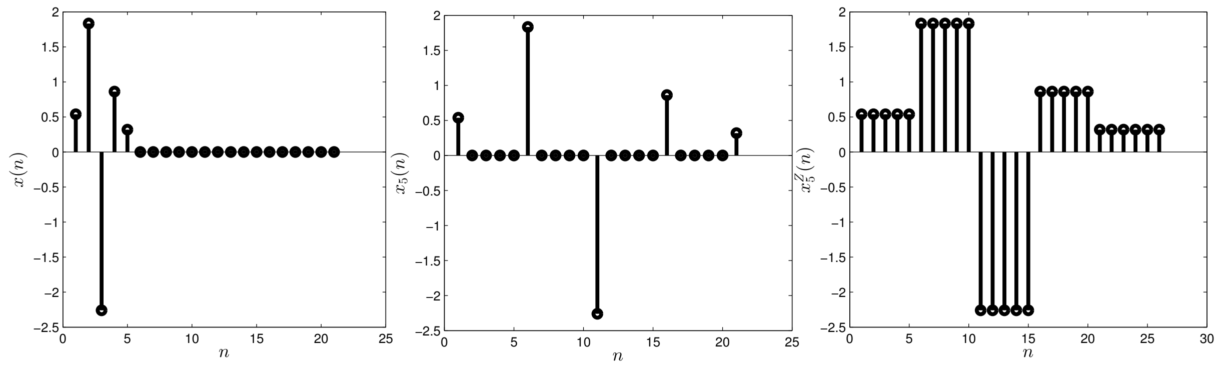

Figure 3: Depiction of ups-ampling and zero-order-hold interpolation filtering.

Zooming in or resizing an image means that we need to increase the number of pixels. Viewing the image as a 2-D signal, resizing the image is equivalent to up-sampling. But how to fill in colors in the extra pixels of the enlarged image? This is done simply by interpolation filters!

Answer the following questions and include all the answers in your report:

1. Up-sampling an image whose pixels are stored in a matrix A, with a factor of L, is equivalent to inserting $L-1$ rows of zeros between every two rows in A and $L-1$ columns of zeros between every two columns in A. For instance, if A is a 3×3 image

whose pixel values are

$$A = \begin{bmatrix} 2 & 6 & 3 \\ 1 & 8 & 5 \\ 7 & 3 & 4 \end{bmatrix},$$

then an up-sampled version of A with a factor of 3 is

$$A_3 = \begin{bmatrix} 2 & 0 & 0 & 6 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 8 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Write a MATLAB command to up-sample an image with an arbitrary factor L. Apply this function to "cameraman.tif" with $L = 3$, and display the resulting image using "imshow". What is the size of the up-sampled image? (Hint: You will need to use the "upsample" command in MATLAB twice, and apply a transpose operation to the image matrix) **(1.5 points)**

2. In folder "Project.rar", a function "ZOH filter" is provided. This function applies a zero-order-hold filter to improve the quality of the up-sampled image. Apply this function (with $L = 3$) to the "cameraman.tif" image and display the result. Comment on the quality of the displayed image compared to what you have obtained in (1). **(1 points)**

3. Other interpolation methods include bilinear and bicubic interpolation. Use "imresize" command (with $L = 3$) to apply both bilinear and bicubic interpolation to "cameraman.tif", and display the output images. Comment on the quality of the output images as compared to those obtained in (1) and (2). **(1 points)**

Part IV: Image Compression using the Discrete Cosine Transform (DCT) (3.5 points)

Can we compress image files into fewer Bytes? Can we represent the same photo in fewer pixels? Yes! Discrete cosine transform (DCT) is a variant of the DFT that is frequently used for the lossy compression of audio (e.g. MP3) and images (e.g. JPEG). The general DCT-based compression algorithm represents the image in the frequency domain in terms of the DCT coefficients, and compresses the image by discarding the high-frequency components since most information are usually centred in the low frequency components.

For 2-D signals, such as images, the DCT transform equation is given by

$$\begin{aligned} X(u,v) &= \sum_{n=0}^{N}\left(\sum_{m=0}^{M} x(n,m)\cos\left[\frac{\pi}{M}\left(m+\frac{1}{2}\right)v\right]\right)\cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)u\right] \\ &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x(n,m)\cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{M}\left(m+\frac{1}{2}\right)v\right], \end{aligned}$$

where $x(n,m)$ is the matrix comprising image pixels, and $X(u,v)$ is the matrix of DCT coefficients. In this exercise, we will see how representing an image using its DCT coefficients can facilitate image compression.

Answer the following questions and include all the answers in your report:

1. Using the "dct2" command, obtain the DCT coefficients of "cameraman.tif". Display the DCT coefficients using the "imshow" command. Comment on the displayed image. **(1 points)**

2. Compression is achieved by gathering the significant DCT coefficients only (usually low frequency components) and discarding other coefficients (usually high frequency components), and then applying inverse DCT. Let A be the matrix in which you store the DCT coefficients of "cameraman.tif". Use "imshow" to display the inverse DCT of both A(1:200,1:200) and A(57:end,57:end) (The inverse DCT is obtained using the "idct2" command). What are the sizes of the two images? Which of the two images is closer to "cameraman.tif" and why? **(1 points)**

3. The *compression ratio* is a measure for the extent to which the image is compressed and is defined as

$$\text{compression ratio} = \frac{\text{Uncompressed image size}}{\text{Compressed image size}}.$$

   Let A be the matrix in which you store the DCT coefficients of "cameraman.tif". Use "idct2" to retrieve the image from A(1:50,1:50), A(1:150,1:150) and A(1:200,1:200). For the three output images, compute the compression ratio and comment on their qualities. What are the trade-offs that one needs to consider when compressing an image? i.e. what do we lose by increasing the compression ratio? **(1.5 points)**