MATLAB for image processing --- A guide to basic MATLAB functions for image processing with MATLAB exercises

Yao Wang and Fanyi Duanmu
Tandon School of Engineering, New York University
Jan. 2016

I Learn about generic tools in Matlab for image processing

I.1 Finding Matlab functions for Image Processing

To determine if the Image Processing Toolbox is already installed on your system, and all the functions provided by the toolbox, type:

```
help images
```

at the MATLAB prompt. If the toolbox is installed, MATLAB responds with a list of image processing functions. if it doesn't, tell your lab assistant so that the problem can be corrected.

I.2 Importing and Exporting Images

Before moving ahead with any image processing operations we need to discuss the process of importing and exporting images to/from the MATLAB environment.

MATLAB 5.3 supports several image file formats, including JPEG (JPG), TIFF,BMP, etc. You can use '`IMREAD`' to read in any image file with a supported format. Use '`help imread`' to find out more details.

To read a gray scale or true color image in one of the supported format, you use

```
[A] = IMREAD(FILENAME,FMT)
```

which will read the image data in FILENAME into a matrix A. If the image file contains a gray scale image, then A is a 2D matrix. If the file contains a RGB color image, then A is a 3D matrix.

For example, if you have a color image in JPEG format, 'image.jpg', with size MxN then using

```
[A]=IMREAD('imag.jpg','JPG')
```

will create a MxNx3 matrix `A`, with `A(1:M,1:N,1)` storing the red component, `A(1:M,1:N,2)` the green component, and `A(1:M,1:N,3)` the blue component.

If the image is in an indexed format, then use

```
[A,MAP] = IMREAD(FILENAME,FMT)
```

A will store the index image, and MAP will store the colormap.

The function IMWRITE allows you to write an image saved in a data matrix to a file with a specified format. For example,

```
IMWRITE(A,'outimg.jpg','JPG')
```

will save the image data A in a file 'outname.jpg' using the JPEG compression format. Use 'help IMWRITE' to find more details.

I.3 Displaying Images

The function imshow displays any supported image. For indexed images, imshow installs the image into the current axes using a specified colormap:

```
imshow(X, map)
```

With no colormap name, imshow uses the current colormap. For the purpose of web publishing we always deal with indexed color images.

We have provided 3 files (JPG, BMP, TIFF) for you to read and Display using MATLAB. You are also welcome to use any image of your own. You must read, display and write an image of each format and become familiar with the process.

To enable you to read and display a gray scale image saved in the raw format, i.e., it contains the pixel value sequentially row by row, with one byte/pixel, a matlab script "h:\el593\exp5\show.m" is provided. For example, to read the raw image "barb.img", you can use

```
show('barb.img',512,512)
```

II Experiment

II.1 MATLAB Warm-up Exercises (Basic Image I/O and Matrix Manipulation)

II.1.1 Read and display a color image.

Use help to understand how "imread" and "imshow" works.
Read in an RGB image, and display it using imshow. Using the "Data cursor" tool under "Tools" in the display window, pick a few points and find out their location and R,G,B values.

Read in the index map image, "rgbadd.gif", save both the index image and the colormap, and display it using imshow. Using the tools->Data cursor tool, pick a few points and find out their location and R,G,B values. What are the RGB values reported for pixels that are in the red, green, blue, magenta, yellow, cyan, and white regions?

Sample Code as follows:

```
[Matrix Map] = imread('Lena.bmp');
```

```
imshow(Matrix, Map);
```

II.1.2 Stitching up multiple images using provided programs

In this task, you should go through the following sample program 'ImageStitching.m', which read in 4 color images, put them into another big image, display and save the resulting image. This example program serves to help you familiarize with the basic MATLAB tools for image I/O and submatrix access. Please apply this program to any 4 images you may have, or use the provided sample images.

Sample Code as follows:

```
% Read in input sub-images
ul = imread('Lena_Upper_Left.jpg');
ur = imread('Lena_Upper_Right.jpg');
ll = imread('Lena_Lower_Left.jpg');
lr = imread('Lena_Lower_Right.jpg');

%  Initialize a image buffer for final stitched image.
[Row Col Layer] = size(ul);
Output = zeros(Row * 2, Col * 2, Layer);

%  Fill in four quadrants with sub-images (by assigning temporary
matrices to target area of final image).
Output(1:Row,1:Row,:) = ul;
Output(1:Row,Row+1:2*Row,:) = ur;
Output(Row+1:2*Row,1:Row,:) = ll;
Output(Row+1:2*Row,Row+1:2*Row,:) = lr;

% Display Sub-images and Stitched Image
subplot(2,2,1); imshow(ul); subplot(2,2,2); imshow(ur);
subplot(2,2,3); imshow(ll); subplot(2,2,4); imshow(lr);
figure; imshow(uint8(Output));title('Stitched Image');

%  Save the final stitched image as a "barbara.bmp" file for later use.
imwrite(Output, 'Lena.bmp');
```

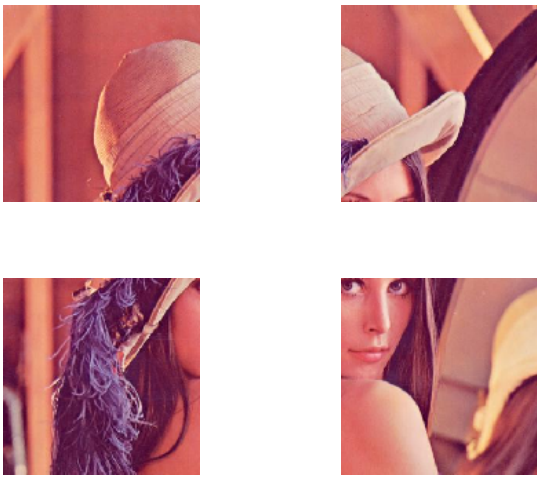

Fig 1 Input Sub-Images

Fig 2 Output Stitched Image

II.1.3 Creating a jigsaw version of an original image

Sample Code as follows:

```
% Read in Image
Input = imread('Lena.bmp');

% Get Height, Width and Component Number
[Row Col Layer] = size(Input); % Row = Col = 512, Layer = 3 -> RGB
```

```
% Calculate Patch Size
Sub_Image_Height = Row / 4;
Sub_Image_Width = Col / 4;

% Create Output Buffer, which is initialized the same as input
Output = Input;

% Misplace some patches
% For simplicity, I move the lower right corner to upper left corner
% You may do something different
Output(1:Sub_Image_Height,1:Sub_Image_Width,:) = Input (Row-
Sub_Image_Height+1:Row,Col-Sub_Image_Width+1:Col,:);

Output(Row-Sub_Image_Height+1:Row,Col-Sub_Image_Width+1:Col,:) =
Input(1:Sub_Image_Height,1:Sub_Image_Width,:);

% Display Output Jigsaw Image
imshow(Output);

% Save Output Image
imwrite(Output, 'Lena_jigsaw.jpg');

% Sometimes your input signal is not between 0 and 255, say, between 0
and 1 or 0 and 512, You may scale your input to 0-255 and then display
it. Sample code as follows
Pixel_max = max(max(Input)); % Find extreme value in your image
Output = floor(Input * 255 / Pixel_max);% Scale Input Signal onto 0-255
```

Fig 3 Output JigSaw Image by Interchanging Two Image Patches

II.1.4 Image grayscale manipulation

Write a program which does the following: 1) read in a color image; 2) Generate a grayscale version of an original RGB image (you can use rgb2gray( ) function of MATLAB); 3) Generate an inverted grayscale image. This can be done by using Img2=255-Img1, if Img1 is a gray scale image in the range o 0-255. 4) Create a binary (black and white) image by thresholding the original grayscale image using a prescribed threshold T so that an original gray value V is change to 0 if V is below T, and is changed to 255 if V is above T.

Hint: The last step can be easily accomplished in MATLAB by Bimg = (Img > T)*255;

II.2 Color model conversion

II.2.1 RGB to YCbCr Conversion

In this part, you should write a program for converting an input image in RGB color model to an image in using the YCbCr model. The program should have the following steps:

1) read in an RGB image based on the given filename and save the image data in a matrix.
2) Display the color image directly, as well display the individual red, green, and blue channels as gray scale image.
3) Call your own function myrgb2ycbcr( ). It should have the syntax: YCCimg=myrgb2ycbcr(RGBimg)
4) Display the Y, Cb, and Cr channels as grayscale images.
5) Call you own function myycbcr2rgb( ). It should have the syntax: RGBimg=myycbcr2rgb2(YCCimg)
6) Display the resulting RGBimg and compare with the original color image.

You should write your own functions YCCimg=myrgb2ycbcr(RGBimg) and RGBimg=myycbcr2rgb2(YCCimg). Your function YCCimg=myrgb2ycbcr(RGBimg) may look something like the following:

```
YCCimg=myrgb2ycbcr(RGBimg)

Rimg=RGBimg(:,:,1);
Gimg=….
Bimg=….
Yimg=zeros( …);
Cbimg=…
Crimg=…
Yimg=floor(0.257*Rimg+0.504*Gimg+0.098*Bimg+16);
Cbimg=…
Crimg=…
YCCimg = zeros(size(RGBimg));
YCCimg(:,:,1) = Yimg;
YCCimg(:,:,2) = Cbimg;
YCCimg(:,:,3) = Crimg;
```

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

Note that you do not look through each pixel. Because the operation for each pixel is the same, you can take advantage MATLAB's built-in capability to do arithmetic on the matrices directly. To see how much time you save, please also run another version of the program, which change the shaded lines in the previous example to

```
(height,width)=size(Rimg);
For (i=1:height, j=1:width)
        Yimg(I,j_) =floor(0.257*Rimg(I,j)+0.504*Gimg(I,j)+0.098*Bimg(I,j)+16);
        Cbimg(I,j)=…
        Crimg(I,j)=…
end
```

Try both versions and see how much different their execution time are.

Please also compare with the result you obtain with Matlab built-in function "rgb2ycbcr()" and "ycbcr2rgb( )".Your report should include your main program and the function. You should also include all the images generated and your comments. You should also comment on the difference of execution time using different versions of the functions as explained above.

II.2.2 RGB to HSV Conversion

In this part, you should write a program to make use of the MATLAB function rgb2hsv( ) to convert a chosen RGB image to the HSV format, and then display the H, S, V channels separately as gray scale images. Please make comments about the correspondence between the color of a pixel and its hue, saturation and value. Include your program, the images generated, and your comments in your report.

II.3 Color Quantization

II.3.1 Uniform quantization on RGB

In this part, you should write a function to apply uniform quantization on each color channels of a RGB image. Your program should have the following syntax:

uniformQuant(infile, outfile, Rlevel,Glevel,Blevel)

Rlevel, Glevel, Blevel should be the number of quantization levels in the R,G,B components.

Your function may contain the following steps:

1) Read in the image from infile.
2) Save red, green, and blue channels into three separate matrices.
3) Determine the quantization stepsize based on the prescribed level for each component.
4) Quantize each color component and return that quantized component (not the index, but the actual quantized value).
5) Combine the quantized color components to a new color RGB image.
6) Display the quantized image
7) Save the quantized image in outfile

Hint: quantizing a single color component Img (assuming minimal value is 0) with stepsize Q can be accomplished by

Qimg=floor(Img/Q)*Q+Q/2

Note that this program will directly save the quantized image as a RGB image, rather than saving the quantized index image and the color map obtained. You are welcome to try to write a program to do that. It is optional.

Use your program with quantization levels equal to 4 and8 for each color component (i.e. using the same levels for all components). Also try to use 16 levels for Green, and 8 for Red and 4 levels for Blue. In each case, determine the total number of color levels and the number of bits per pixel needed to represent the quantized color index image. Are there two settings with the same total number of bits per pixel? How do their quality compare?

II.3.2 Uniform quantization on YCbCr

This time, you firstly convert input RGB image into YCrCb format. You can use your previously written function myrgb2ycrcr for the conversion. If that did not work well, you could also use MATLAB function rgb2ycbcr( ). Then you apply uniform quantization on Y, Cr and Cb components using prescribed number of levels, and then converted it back to RGB color domain for display.

Use your program with quantization levels equal to8 for each color component (i.e. using the same levels for all components). Also try to use 16 levels for Y, and 8 for Cb and 4 levels for Cr. Also try to use 32levels for Y, and 4 for Cb and 4 levels for Cr. How do their quality compare? Also, compare with the quality when the image is quantized in the RGB domain directly with 8 levels for each component, and with 16, 8, 4 levels for R,G, B, respectively. How do these cases differ in the number of bits per pixel needed to represent the quantized image?

II.3.3 Uniform and Adaptive quantization using MATLAB function rgb2ind

Use help to understand the syntax needed to use rgb2ind to implement uniform and adaptive quantization (known as minimal variance quantization in MATLAB), and to enable or disable dithering option.

Use rgb2ind() to quantize the same RGB image you used for task 3.1 with the same total number of color levels, using both uniform and adaptive quantization, with and without dithering. Compare their qualities. Also compare the result by uniform quantization with the ones you obtained in task 3.1.