Loading and Displaying Images in Matlab

The main aim of this set of notes is to learn how to read, display, and save images in Matlab. Before we begin, we should be aware of basic terms used in the computer manipulation of images.

1 Some Image Basics

There are a variety of *image data models* used in digital image processing. In addition, there are a variety of *image file formats* used by computers.

1.1 Image Models

Images can be color, grayscale, or black and white. Color images can use different color models, such as RGB, HSV and CMY. For our purposes, we will be mainly concerned with grayscale (sometimes called intensity) images.

An intensity image can be thought of simply as a 2-D array (or matrix), where each entry contains the intensity value of the corresponding pixel. Typical grayscales for intensity images can have integer values from $[0, 255]$ or $[0, 65535]$, where 0 is black, and the right endpoint (255 or 65535) is white.

In addition, Matlab supports double precision floating point numbers for intensity values of images (these are in the interval $[0, 1]$, where 0 is black and 1 is white), but most often we will be reading in image files that have been stored as integers. We must be aware of the various data types, and make sure we convert the data (if necessary); we'll discuss this point later.

1.2 Image File Formats

There are many types of image file formats that are used to store images. Currently, the most commonly used formats include:

- GIF (Graphics Interchange Format)
- JPEG (Joint Photographic Experts Group)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)

The development of PNG was motivated by legal problems relating to the patented compression algorithm used in GIF images. In fact, Matlab used to be able to read and write GIF images, but because of legal problems, they had to remove those functions. It is likely that the PNG format will continue to become increasingly more popular. Because Matlab supports many different file formats, we can use it to convert from one to another (for example, read a tiff file, and write it as a png file).

2 Some Matlab Basics

Here we describe some basics of how to read and display images in Matlab. Keep in mind that you can find more detailed information on any Matlab command discussed in this section by using the `help` command. In addition, `helpdesk` has a lot of good information on using Matlab's Image Processing Toolbox.

2.1 Test Images

The Image Processing Toolbox contains a bunch of test images which, on the Math/CS system, are in the directory

```
/usr/local/matlab-r11/toolbox/images/imdemos/
```

You can see these files by entering the command in the Matlab command window:

```
>>!ls /usr/local/matlab-r11/toolbox/images/imdemos/
```

Notice there are several tiff files. Any of these images can be used in what is discussed below.

2.2 Information About Image Files

The command `imfinfo` displays information about the image stored in a data file. For example,

```
>> info = imfinfo('flowers.tif')
```

shows that the image contained in the file `flowers.tif` is an RGB image. Doing the same thing for `saturn.tif`, we see that this image is a grayscale intensity image.

2.3 Reading Images

The command to read images in Matlab is: `imread`. The help command tells more about how to use this, but here are two examples:

```
>> F = imread('flowers.tif');
>> S = imread('saturn.tif');
```

Now use the `whos` command to see what variables you have in your workspace. Notice that both `F` and `S` are arrays whose entries are "uint8" values. This means the intensity values are in the range $[0, 255]$.

2.4 Displaying Images

There are two basic commands for displaying images: `imshow` and `imagesc`. In general, `imshow` is the preferred command, since it renders the image more accurately, especially in terms of size, and for color images. However, in some cases `imagesc` works better for grayscale images. Try:

```
>> figure(1), imshow(F)
>> figure(2), imagesc(F)
>> figure(1), imshow(S)
>> figure(2), imagesc(S)
```

Rather than putting images in different figure windows, we might want to put all images in the same figure, side by side, or one above the other. To do this, we can use the `subplot` command as follows:

```
>> close all
>> figure
>> subplot(1,2,1), imshow(F)
>> subplot(1,2,2), imshow(S)
```

Problem 1 *What commands would you use to put one image above another? How about four images in the same figure?*

2.5 Writing Images

To write an image to a file using any of the supported formats we can use the `imwrite` command. There are many ways to use this function, and you should see the on-line help for more information. Here we only describe two basic approaches, which will work for converting images from one data format to another; for example, from tiff to jpeg. This can be done simply by using `imread` to read an image of one format, and `imwrite` to write it to a file of another format. For example:

```
>> I = imread('image.tif');
>> imwrite(I, 'image.jpg');
```

3 Manipulating Images in Matlab

We have learned the very basics of reading and writing images in Matlab. We now some basics of performing arithmetical operations on images. One important thing we have to keep in mind is that most image processing software expects the pixel values (entries in the image arrays) to be in a fixed interval. Recall that typical grayscales for intensity images can have integer values from $[0, 255]$ or $[0, 65535]$ or $[0, 1]$, where 0 is black, and the right endpoint (255 or 65535 or 1) is white.

If, after performing some mathematical operations, the pixel values fall outside these intervals, unexpected results can occur. Moreover, since our goal is to operate on images with sophisticated mathematical methods, integer representation of images can be limiting. For example, if we multiply an image by a noninteger scalar, then the result contains entries which are nonintegers. Of course, we can easily convert these to integers by, say, rounding. If we're only doing one mathematical operation, then this approach is fine. However, some manipulations we consider later require doing many, many mathematical operations, and rounding, chopping and rescaling after each one of these can lead to significant loss of information. Therefore, we would like to have "images" which contain entries that are double precision, floating point numbers, and not necessarily restrict ourselves to a fixed interval. After our algorithms are finished processing the results, and we are ready to display or write them as images, then we can convert back to an appropriate format.

3.1 Basic Mathematical Operations

Let's start off doing something we know: reading and displaying an image:

```
>> S = imread('saturn.tif');
>> imshow( S )
```

For us, the next most important topic is to understand how to algebraically manipulate the images; that is, we want to be able to add, subtract, multiply and divide images. Unfortunately, standard MATLAB commands such as +, -, *, /, do not always work for images. For example, you might try:

```
>> S + 10
```

The + operator does not work for uint8 variables! Unfortunately, most images stored as tiff, jpeg, etc., are either uint8 or uint16, and standard math operations don't work on these types of variables.

Although MATLAB has functions such as `imadd, imsubtract, immultiply, imdivide` that can be used for this purpose, we won't use them. Instead, we will convert the image array to a standard matrix, with double precision floating point entries. We will operate on this matrix, and then, if necessary, convert the matrix back to an "image".

Since, in general, we will only consider grayscale intensity images, the main conversion function we need is `double`. It's easy to use:

```
>> Sd = double(S);
```

After running this command, use the `whos` command to see what variables are in the workspace. Notice that `Sd` requires a lot more memory, but now we are not restricted to working only with integers, and standard mathematical operations like +, -, *, / all work.

In some cases, we may want to convert color images to grayscale intensity images. This can be done by using the commands: `ind2gray, rgb2gray`. If we then plan to use mathematical operations on these images, then we will also need to convert to double. For example:

```
>> F = imread('flowers.tif');
>> Fg = rgb2gray(F);
>> Fgd = double(Fg);
```

In general it is not a good idea to change "true color" images to grayscale, since we are loosing information. Mathematical algorithms for manipulating color images typically work on the intensity values of each of the three RGB colors. We will not do anything complicated with color images, and instead stick to grayscale.

Once the image is converted to a double array, we can use any of the array operations on it.

Problem 2 *What is computed by the* MATLAB *statements?*

```
>> size(Sd)
>> max(max(Sd))
>> min(min(Sd))
```

3.2 Displaying Revisited

Now that we are in a position to algebraically manipulate images, we need to see how results are displayed. Note that `Sd` is only using more storage for the entries in the array, but the values are really the same (try looking at the values `Sd(200,200)` and `S(200,200)`). But try to display the image `Sd`:

```
>> imshow(Sd)
```

To understand the problem here, we need to understand how `imshow` works.

- When the input image has uint8 entries, it expects the values to be integers in the range $[0, 255]$. If they are not in this range, truncation is done; in particular, entries less than 0 are set to 0, entries larger than 255 are set to 255. Then the image is displayed.

- When the input image has uint16 entries, it expects the values to be integers in the range $[0, 65535]$. If they are not in this range, truncation is done; in particular, entries less than 0 are set to 0, entries larger than 65535 are set to 65535. Then the image is displayed.

- When the input image has double entries, it expects the values to be (nonintegers) in the range $[0, 1]$. If they are not in this range, truncation is done; in particular, entries less than 0 are set to 0, entries larger than 1 are set to 1. Then the image is displayed.

The array `Sd` has entries that range from 0 to 255, but they are double. So, before displaying the image, all the entries greater than 1 are set to 1. We can get around this situation in two ways. The first is to tell `imshow` that the max and min are different from 0 and 1 as follows:

```
>> imshow(Sd, [0, 255])
```

Of course this means we need to know the max and min values in the array. If we don't know these, and don't want to figure out what they are, we can simply use:

```
>> imshow(Sd, [])
```

In this case, `imshow` finds the max and min values in the array, and scales everything down to $[0, 1]$, and then displays.

The other way to fix this scaling problem, is to rescale `Sd` into an array with entries in $[0, 1]$, and then display. This can be done as follows:

```
>> Sds = mat2gray(Sd);
>> imshow(Sds)
```

Probably the most common way we will use `imshow` is `imshow(Sd,[])`, since it will give consistent results, even if the scaling is already in the interval $[0, 1]$.

Problem 3 *Explain what happens when the following* MATLAB *statement is executed.*

```
>> imshow(max(max(Sd))-Sd,[])
```

That is, explain what happens when each individual operation `max(max(Sd))`*,* `max(max(Sd))-Sd`*, and* `imshow(max(max(Sd))-Sd,[])` *is performed.*

3.3 Writing Revisited

We saw that to write an image to a file using any of the supported formats we can use the `imwrite` command. What we didn't say is that it is very important, before using `imwrite`, to make sure the image is converted (rescaled) properly. In particular, if the image array is of type double, then use the `mat2gray` command.

The two formats we will use most are jpeg and png, which can be used as follows (where we assume `X` is an array of type double, not necessarily converted to grayscale:

```
>> imwrite(mat2gray(X), 'MyImage.jpg', 'Quality', 100)
>> imwrite(mat2gray(X), 'MyImage.png', 'BitDepth', 16, 'SignificantBits', 16)
```

Problem 4 *Consider the previous problem, where you displayed the image* `max(max(Sd))-Sd`*. How would you write this image to a file? You might want to try writing it to a file, and then read it in again and see if you can display it correctly.*

Finally, we remark that if 16 bits is not enough accuracy, and you want to save your images with their full double precision values, then you can simply use the `save` command.