Math 551 Fall 2018
Lab 3

Goals: We will use matrix multiplication in order to manipulate color of images and achieve different color effects.

To get started:

Create a new Matlab script file and save it as "lab03.m".

Download the file "baboon.jpg" and save it in your working directory.

Matlab commands to learn: `imread, imshow, size, figure, reshape, uint8, double, for... end`

What you have to submit: The file `lab03.m`, which you will create during the lab session.

Reminders About the Grading Software

Remember, the labs are graded with the help of software tools. If you do not follow the instructions, you will lose points. If the instructions ask you to assign a value to a variable, you must **use the variable name given in the instructions**. If the instructions ask you to make a variable named `x1` and instead you make a variable named `x` or `X` or `X1` or any name other than `x1`, the grading software will not find your variable and you *will* lose points. Required variables are listed in the lab instructions in a gray box like this:

Variables: `x1`, `A`, `q`

At times you will also be asked to answer questions in a comment in your M-file. You must **format your text answers correctly**. Questions that you must answer are shown in gray boxes in the lab. For example, if you see the instruction

Q7: What does the Matlab command `lu` do?

your file must contain a line similar to the following somewhere

```
% Q7: It computes the LU decomposition of a matrix.
```

The important points about the formatting are

- The answer is on a single line.
- The first characters on the line is the comment character `%`.
- The answer is labeled in the form `Q`n`:`, where n stands for the question number from the gray box.

If you do not format your answers correctly, the grading software will not find them and you *will* lose points.

INTRODUCTION

Have you ever wondered how Instagram color filters work? A color image can be represented by a matrix with the dimensions corresponding to the dimensions of the image. Each of the elements of this matrix contains the number (or numbers) representing the color of the corresponding pixel. If the image is a color image, then the color of the pixel is represented by three numbers {R,G,B} (Red, Green and Blue) with each number ranging from 0 to 255 (RGB system). In this project we will learn to manipulate these colors in order to obtain different color effects.

TASKS

1. Start by loading the image into Matlab and displaying it on the screen by using the commands `imshow` and `imread` (type `help imshow` and `help imread` to learn about these commands.) Save the resulting array as `ImJPG`. You should see the image in the Figure 1.

   Variables: `ImJPG`

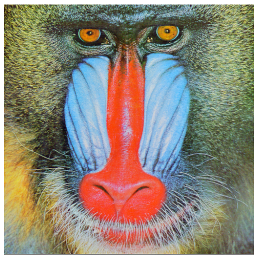

Figure 1: The original image

2. Check the dimensions of the array `ImJPG` using the command `[m,n,l]=size(ImJPG)`. Observe, that Matlab will return three numbers. Since the image is colored, the resulting array is three-dimensional. Among other things, Matlab allows operations on multidimensional arrays. The first two dimensions of the array `ImJPG` correspond to the horizontal and vertical dimensions (number of pixels) of the image, and the third dimension stores three numbers corresponding to the values of Red, Green, and Blue for each pixel. These three colors mixed together produce all other colors in the image. The values for Red, Green and Blue can range from 0 to 255, allowing us to create $256^3 = 16,777,216$ colors. That is more than an ordinary human eye can distinguish. In this project, we will be manipulating these three numbers to achieve various visual effects (similar color filters in Instagram).

   Variables: `m,n,l`

3. First of all, let us look into amount of each of Red, Green, and Blue color in the image. To do this, let us extract individual layers or color channels from the image array. To obtain the red channel of color in the image, use the following command:

   ```
   redChannel = ImJPG(:,:,1);
   ```

   Similarly, extract green and blue color channels and save them in the arrays `greenChannel` and `blueChannel`. Display each array in a separate figure using `figure` and `imshow` commands. Observe that individual color channels will be shown by Matlab as grayscale images. That is due to the fact that we have taken “layers” off of the matrix `ImJPG` and each of these layers individually looks like a two-dimensional array with numbers ranging from 0 to 255 aka like a grayscale image. If you did everything correctly, you should see three images like in Figure 2. The whiter the pixel appears on a particular color channel, the larger amount of that color is contained in that pixel. Other way around, the darker the area is, the less corresponding color is in that part of the image.

Variables: `redChannel, greenChannel, blueChannel`

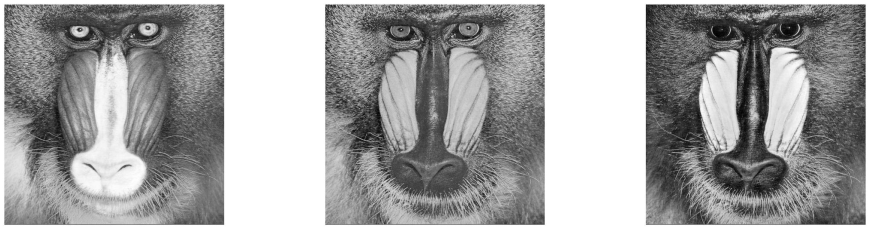

Figure 2: Different color channels

4. Let us convert the original image to a grayscale image by using the following matrix:

$$\texttt{GrayMatrix} = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

This matrix produces an average of red, green, and blue for each pixel. Use the following code:

```
GrayMatrix=[1/3 1/3 1/3; 1/3 1/3 1/3; 1/3 1/3 1/3];
for i = 1:m
      for j = 1:n
           PixelColor = reshape(double(ImJPG(i,j,:)),3,1);
           ImJPG_Gray(i,j,:) = uint8(GrayMatrix*PixelColor);
      end;
end;
figure;
imshow(ImJPG_Gray)
```

This code produces an array `ImJPG_Gray`. Observe the use of the commands `reshape, uint8, double`. For every pixel of the image, first, the color attributes (RGB) are extracted from the matrix ImJPG, then these color attributes are treated as a vector with three components [R,G,B], and finally, the color attributes are changed by multiplying the vector [R, G, B] by the filter matrix `GrayMatrix` on the left. The result is saved then as color attributes of the corresponding pixel in the array `ImJPG_Gray`. All the pixels of the array `ImJPG_Gray` have equal number of Red, Green, and Blue; this produces different shades of gray color. Observe that there are different ways to create a grayscale conversion, and this is just one of them.

Variables: `ImJPG_Gray`

Q1: What do the `uint8` and `double` commands do in the code above?

5. Modify the code above to produce a sepia conversion of the image. Instead of `GrayMatrix` use the

following filter matrix, and reproduce the code above with this matrix:

$$\texttt{SepiaMatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

Save the result in the `ImJPG_Sepia` array and display this array using `imshow` command.

Variables: `ImJPG_Sepia`

6. Next, consider the filter matrix

$$\texttt{RedMatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Modify the code again, using the matrix above as a filter matrix. The result should look like the left image in Figure 3. Save the result as `ImJPG_Red`. Display the image.

Variables: `ImJPG_Red`

Q2: What does the transformation (i.e. multiplying by the matrix `RedMatrix`) do to the image?

7. Now let us delete one of the colors in the image, say red. First, produce the matrix which deletes red color in the image and keeps the values of Green and Blue the same. Use it to filter the image. The result should look like the right image in Figure 3. Save the result as `ImJPG_DeleteRed`.

Variables: `ImJPG_DeleteRed`

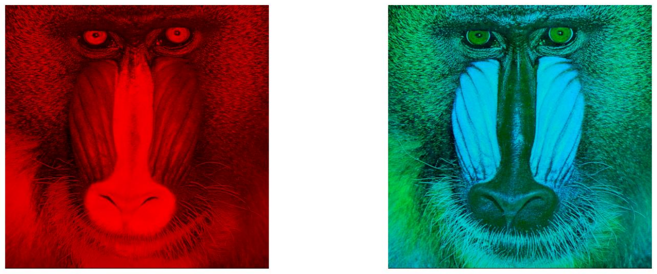

Figure 3: Left: Multiplying by `RedMatrix`. Right: Red color deleted

8. Let us permute the colors in the image. To do this repeat the steps above with the matrix:

$$\texttt{PermuteMatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Save the result as `ImJPG_Permute`. If you did correctly, you should see the image in Figure 4.

Variables: `ImJPG_Permute`

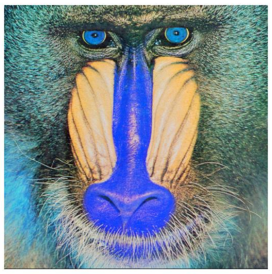

Figure 4: Colors permuted

The matrix `PermuteMatrix` is an example of hue rotation filter. You can produce other hue rotation effects with this more general transformation:

$$\texttt{HueRotationMatrix} = \begin{bmatrix} 0.213 & 0.715 & 0.072 \\ 0.213 & 0.715 & 0.072 \\ 0.213 & 0.715 & 0.072 \end{bmatrix} + \cos(\theta) \begin{bmatrix} 0.787 & -0.715 & -0.072 \\ -0.213 & 0.285 & -0.072 \\ -0.213 & -0.715 & 0.928 \end{bmatrix}$$
$$+ \sin(\theta) \begin{bmatrix} 0.213 & -0.715 & 0.928 \\ 0.143 & 0.140 & -0.283 \\ -0.787 & 0.715 & 0.072 \end{bmatrix}.$$

Here θ is the angle of rotation. You can try experimenting with various values of the angle θ to get different color effects.

9. It is also possible to amplify/deamplify individual colors in the image. For instance, consider the color transformation with the matrix:

$$\texttt{SaturateMatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Modify the code with matrix `SaturateMatrix`. Save the result as `ImJPG_Saturate`.

Variables: `ImJPG_Saturate`

Q3: What does the transformation above do to the image?

10. It is possible to invert the colors of the image by using the following code:

```
ImJPG_Invert=255-ImJPG;
figure;
imshow(ImJPG_Invert);
```

Observe that here again Matlab automatically substitutes matrix of appropriate dimension (in this case $m \times n \times l$) with all the elements equal to 255 instead of the constant 255.

Variables: `ImJPG_Invert`