Image Courtesy: Max Pixel

The R, G, and B in an Image

Yedhu Krishnan · Follow

Published in Analytics Vidhya

5 min read · Oct 8, 2019

If you haven't read the last post in the series, read it here: Introduction to Digital Image Processing in Python.

A typical color image consists of three color channels: red, green and blue. Each color channel has 8 bits and can represent 256 distinct values. Using a combination

of all three channels, we can create `256 x 256 x 256` colors, which is around 16-million colors. You might have heard this term before. Now you know where that came from.

In this post, we will learn more about the different components of an image. Let us try everything in a python console, which can be started by running the following command in a terminal. If you haven't installed Python yet, please follow the instructions from my previous post.

```
python3
```

You can download and use the sample image given below. It has different colors, and that's what we need.

Image Courtesy: Max Pixel

Import the library and load the image:

```
import imageio
image = imageio.imread('colors.jpg')
```

Now we have the image as a NumPy array in our console. If you want to see the dimensions of the image, run:

```
image.shape
```

You will get something like `(426, 640, 3)`. This means the image has a height of 426 pixels, a width of 640 pixels, and three color channels. We are interested in the color channels for now. Let us write the individual channels to separate image files:

```
imageio.imwrite('red.jpg', image[:, :, 0])
imageio.imwrite('green.jpg', image[:, :, 1])
imageio.imwrite('blue.jpg', image[:, :, 2])
```

The `:` is used to get all values. For example, `image[:, :, 0]` means get all rows, all columns, and the first (at index 0) color channel, which is the red channel. I'll talk more about Indexing and Slicing in the next post. If you are curious, you can learn more about here: Indexing and Slicing (or here).

If you open the files you just created, you would see something like this.

The red, blue and green components

Why do they all look grey?

That is because they are single-channel images now. They will be saved as greyscale images. If you see the shape of the green component using the command:

```
image[:, :, 1].shape
```

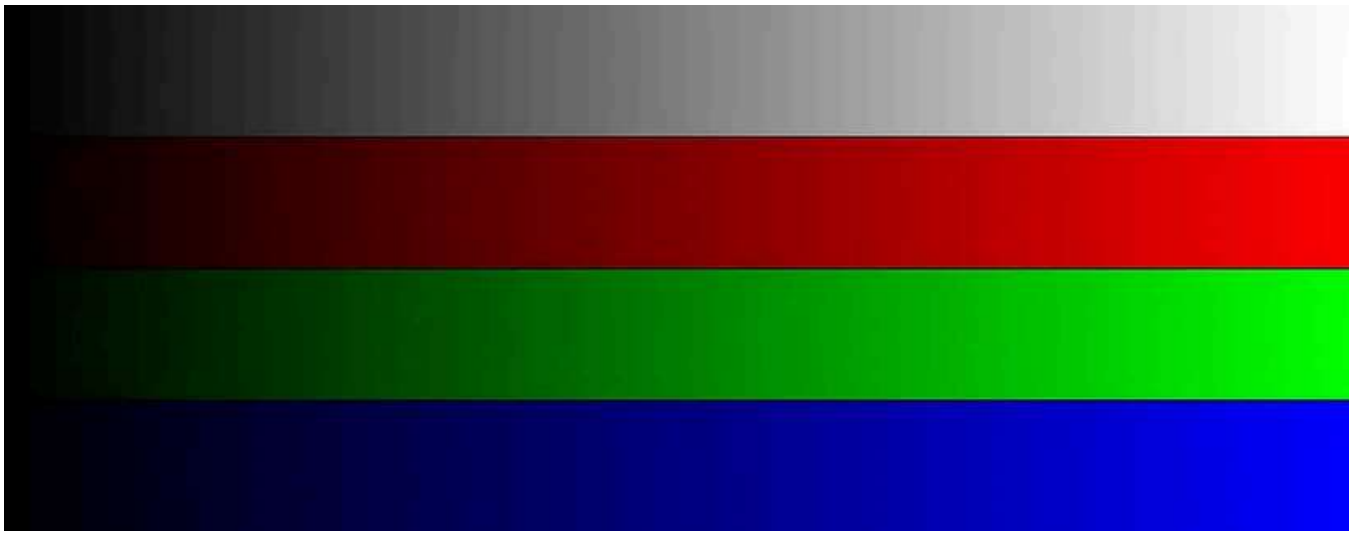
You will see `(426, 460)` . The color channel part is missing. Each pixel will have a single 8-bit value (unlike in RGB, which has three 8-bit values for R, G and B).

But if observe carefully, you will see that the grey value in the red-channel image corresponds to the amount of red in the color image.

The distribution of red color. On the extreme left, the value of red would be 0 and on the extreme right, it would be 255. As we move from left to write, the value gradually increases.

For example, see a red straw in the color image. A red pixel would be `[255, 0, 0]` , which is on the right side of the color distribution image shown above. The green and blue components will be zero.

In `red.jpg` image, the color of the same straw is white, which is 255. And the same straw appears black (`0`) in both green and blue component images because a red pixel doesn't have blue or green components.



Color distribution for greyscale, red, green and blue channels. In all cases, the left side is 0 and the right side is 255.

If you want to see the actual colors instead of just grey images, you need to retain all three channels and set the values on the other channels to zero. For example, to get the red component, set the green and blue channel values to 0, as shown below:

```
red_image = image.copy()
red_image[:, :, 1] = 0
```

```
red_image[:, :, 2] = 0
imageio.imwrite('red_image.jpg', red_image)
```

Here, I made a copy of the image and set all the blue and green channel values to zero. This is another handy usage of `:` to set the same value to multiple pixels in a NumPy array.

Similarly, you can create the green and blue images. They will now look like this:

The colorful red, green and blue components

Notice that for yellow straws, both red and green components are active, but the blue component is missing (black in color). The pixel values would be closer to `[255, 255, 0]`.

White color in RGB image would be represented as `[255, 255, 255]`. Grey pixels in the RGB image will have the same value for all components. When we take the average of all three values, we get the same number. For example, a grey value `[57, 57, 57]` in RGB will have value `57` in a greyscale image. We discussed about converting an RGB image to greyscale in the first post.

Now you know how RGB images are made; by using a combination of values on R, G, and B color channels. The same technique is used in all modern color displays, including mobile and TV screens. Here is a cool video by The Slow Mo Guys explaining how that works on TVs.

In the coming posts, I'll explain about processing a digital image using Python. For now, please share your feedback and suggestions.