1.4.9: Splitting and merging channels

Topic Progress: (https://gurus.pyimagesearch.com/topic/translation/) (https://gurus.pyimagesearch.com/topic/rotation/) (https://gurus.pyimagesearch.com/topic/resizing/) (https://gurus.pyimagesearch.com/topic/flipping/) (https://gurus.pyimagesearch.com/topic/cropping/) (https://gurus.pyimagesearch.com/topic/image-arithmetic/) (https://gurus.pyimagesearch.com/topic/bitwise-operations/) (https://gurus.pyimagesearch.com/topic/masking/) (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/)

As we know, an image is represented by three components: a Red, Green, and Blue channel. And while we've (briefly) discussed grayscale and binary representations of an image, you may be wondering: "*How do I access each* ***individual*** *Red, Green, and Blue channel of an image?*"

Since images in OpenCV are internally represented as NumPy arrays, accessing each individual channel can be accomplished in multiple ways, implying that there are multiple ways to skin this cat. However, we'll be focusing on the two main methods that you should be using: `cv2.split` and `cv2.merge` .

Objectives:

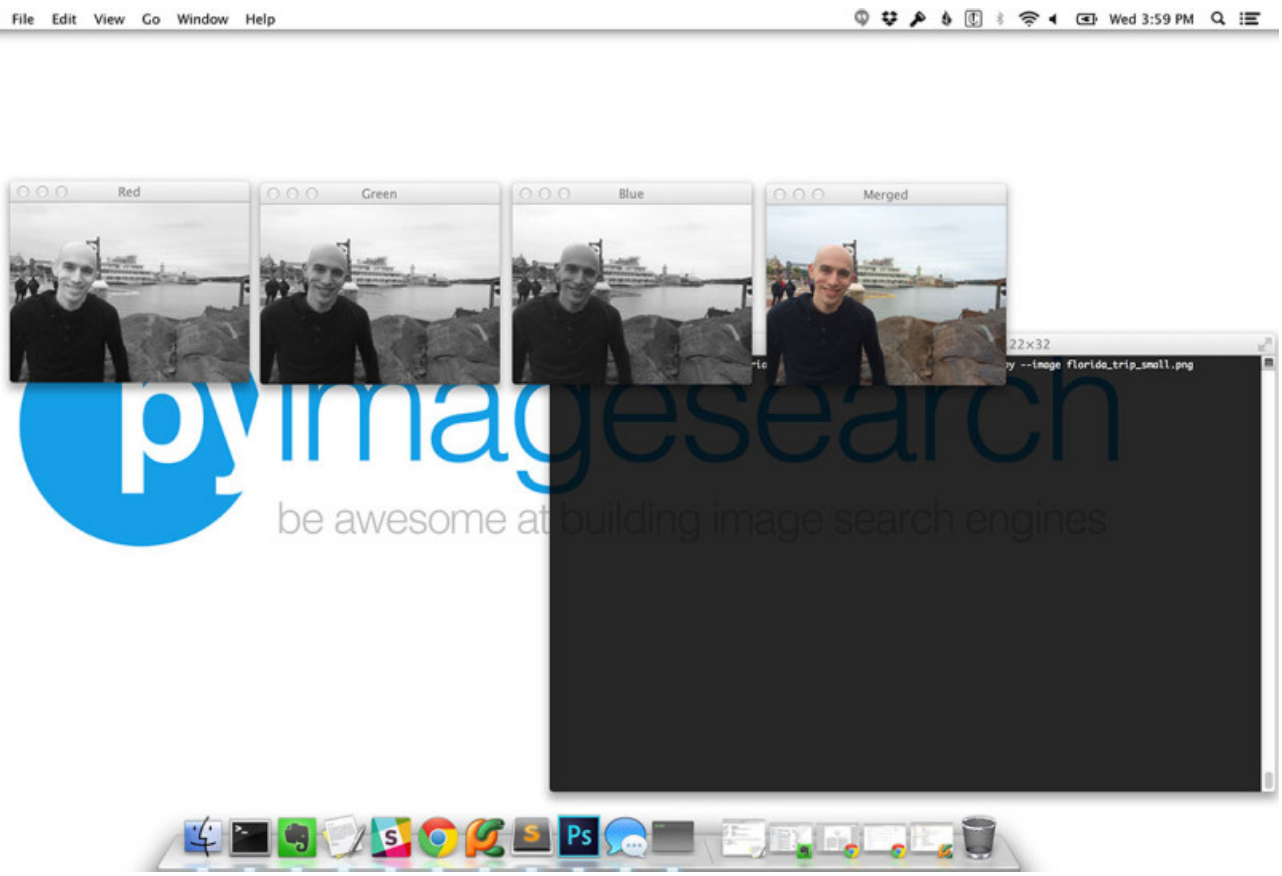
By the end of this topic you should understand how to both *split* and *merge* channels of an image by using the `cv2.split` and `cv2.merge` functions.

Splitting and Merging Channels

A color image consists of multiple channels: a Red, a Green, and a Blue component. We have seen that we can access these components via indexing into NumPy arrays. But what if we wanted to split an image into its respective components?

As you'll see, we'll make use of the `cv2.split` function.

But for the time being, let's take a look at an example image in **Figure 1:**



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/color_channels.jpg)

FIGURE 1: THE THREE RED, GREEN, AND BLUE CHANNELS, RESPECTIVELY, OF OUR IMAGE, FOLLOWED BY THE ORIGINAL IMAGE ITSELF.

Here we have (in the order of appearance) the Red, Green, Blue, and original image of myself on a trip to Florida.

But given these representations, how do we interpret the different channels of the image?

Well, let's take a look at the color of the sky in the *original* image. Notice how the sky is a slightly *blue* tinge. And when we look at the blue channel image we see that the blue channel is *very light* in the region that corresponds to the sky. This is because the blue channel pixels are very bright, indicating that they

contribute heavily to the output image.

Then, take a look at the black hoodie that I am wearing. In each of the Red, Green, and Blue channels of the image my black hoodie is *very dark* — indicating that each of these channels contribute very little to the hoodie region of the output image (giving it a very dark black color).

When you investigate each channel *individually* rather than as a *whole*, you can visualize how much each channel contributes to the overall output image. Performing this exercise is extremely helpful, especially when you are applying methods such as thresholding and edge detection, which we'll cover later in this module.

Now that we have visualized our channels, let's examine some code to accomplish this for us:

splitting_and_merging.py Python

```
1  # import the necessary packages
2  import numpy as np
3  import argparse
4  import cv2
5
6  # construct the argument parser and parse the arguments
7  ap = argparse.ArgumentParser()
8  ap.add_argument("-i", "--image", required=True, help="Path to the image")
9  args = vars(ap.parse_args())
10
11 # Load the image and grab each channel: Red, Green, and Blue. It's
12 # important to note that OpenCV stores an image as NumPy array with
13 # its channels in reverse order! When we call cv2.split, we are
14 # actually getting the channels as Blue, Green, Red!
15 image = cv2.imread(args["image"])
16 (B, G, R) = cv2.split(image)
17
18 # show each channel individually
19 cv2.imshow("Red", R)
20 cv2.imshow("Green", G)
21 cv2.imshow("Blue", B)
22 cv2.waitKey(0)
23
24 # merge the image back together again
25 merged = cv2.merge([B, G, R])
26 cv2.imshow("Merged", merged)
27 cv2.waitKey(0)
28 cv2.destroyAllWindows()
```

Lines 1-14 imports our packages, sets up our argument parser, and then loads our image. Splitting the channels is done using a call to `cv2.split` on **Line 16**.

Normally, we think of images in the RGB color space — the red pixel first, the green pixel second, and the blue pixel third. However, OpenCV stores RGB images as NumPy arrays in reverse channel order.

Feedback

Instead of storing an image in RGB order, it stores the image in BGR order. Thus we unpack the tuple in reverse order.

Lines 19-22 then show each channel individually, as in **Figure 1** above.

We can also merge the channels back together again using the `cv2.merge` function. We simply specify our channels, again in BGR order, and then `cv2.merge` takes care of the rest for us (**Line 25**)!

In fact, the output of **Line 25** is the image on the *far right* in **Figure 1** — notice how we were able to reconstruct the original image that we loaded off disk on **Line 15** using the `cv2.merge` function. This is an especially useful trick, especially if we need to process each channel of the image individually.

There is also a second method to visualize the color contribution of each channel. In **Figure 1** we were simply examining the single channel representation of an image, which to us looks like a grayscale image. However, we can also visualize the color contribution of the image as a full RGB image, like this:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/color_channels_visualized.jpg)

FIGURE 2: A SECOND METHOD TO VISUALIZING THE COLOR CONTRIBUTION OF EACH CHANNEL.

Using this method we can visualize each channel in "color" rather than "grayscale." In general, this is *strictly* a visualization technique and not something we would use in a normal computer vision or

image processing application. But that said, let's investigate the code to see how to construct this representation:

splitting_and_merging.py Python

```
30 # visualize each channel in color
31 zeros = np.zeros(image.shape[:2], dtype = "uint8")
32 cv2.imshow("Red", cv2.merge([zeros, zeros, R]))
33 cv2.imshow("Green", cv2.merge([zeros, G, zeros]))
34 cv2.imshow("Blue", cv2.merge([B, zeros, zeros]))
35 cv2.waitKey(0)
```

In order to show the actual "color" of the channel, we first need to take apart the image using `cv2.split` . Then we need to reconstruct the image, but this time, having all pixels *but the current channel* as zero.

On **Line 31** we construct a NumPy array of zeros, with the same width and height as our original image. Then, in order to construct the Red channel representation of the image, we make a call to `cv2.merge` , but specifying our `zeros` array for the Green and Blue channels. We take similar approaches to the other channels in **Line 33 and 34**.

Summary

In this topic we explored how to utilize the `cv2.split` and `cv2.merge` functions to access each of the individual RGB channels of an image. Since images in OpenCV are represented as NumPy arrays, there are multiple ways to access each channel, but using `cv2.split` and `cv2.merge` tend to be the easiest (and furthermore, most relevant to the OpenCV library in this course).

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/splitting_and_merging.zip)

Quizzes		Status
1	Splitting and Merging Quiz (https://gurus.pyimagesearch.com/quizzes/splitting-and-merging-quiz/)	

Feedback

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback