CHAPTER 10

Getting Started with Scikit-Image

In the last two chapters, we got started with image processing with **NumPy** and **matplotlib**. In this chapter, we will have an introduction to scikit-image library which is a dedicated library for image processing and computer vision. We will have an introduction to the scikit project too. We will learn how to setup Windows PC and Raspberry Pi for image processing with scikit-image. Then, we will have a look at a few simple operations offered by the library.

10.1 Introduction to Scikits

Scikits stand for SciPy toolkits. These libraries are not the part of core **SciPy** library. They are hosted, developed, and maintained independently from core SciPy libraries. You can find out more about **scikits** at https://www.scipy.org/scikits.html and http://scikits.appspot.com/scikits. **Scikit-image** is a scikit and it specializes in image processing and computer vision. All the scikits heavily use **NumPy** and **SciPy** for implementation of various functionalities.

10.2 Installation of Scikit-learn on Windows and Raspberry pi Raspbian

Scikit-image requires cython package and cython requires a C++ compiler. For windows, we need to install the latest version **Microsoft Visual C++ Redistributable**. We can find it at https://visualstudio.microsoft.com/downloads. Download the setup file appropriate for your Windows (32 bit or 64 bit). Install the **VC++** redistributable by executing the setup file. Once done, open the command prompt and run the following command,

```
pip3 install scipy cython scikit-image
```

Raspberry Pi Raspbian and the other distributions of Linux already have gcc for C++. So, we can directly install required libraries. Just run the following command in terminal of Raspberry Pi,

```
sudo pip3 install scipy cython scikit-image
```

10.3 Basics of Scikit-image

Now, we will see the basics of **scikit-image**. Just like **matplotlib**, skikit-image has function imread(), imshow(), and show() to read and display images. They work exactly same as their counterparts in **matplotlib**. Following is the sample program to demonstrate the functionality of these functions,

```
%matplotlib inline
import skimage.io as io
img = io.imread('D:\\Dataset\\4.2.05.tiff')
print(type(img))
io.imshow(img)
io.show()
```

The code above reads and displays an image. We are using `io` module of scikit-image for that. In this section, we will go through this module in detail.

Scikit-image has many images in the data module which we can use to demonstrate image processing operations. We can use the images in the `data` module as follows,

```
import skimage.data as data
img = data.astronaut()
io.imshow(img)
io.show()
```

The code above shows the image of astronaut **Eileen Collins** (https://en.wikipedia.org/wiki/Eileen_Collins), a retired **NASA** astronaut.

We can use `imread()`, `imshow()`, and `show()` from **matplotlib** too to display images as follows,

```
import matplotlib.pyplot as plt
img = data.coffee()
plt.imshow(img)
plt.title('Coffee')
```

```
plt.axis('off')
plt.show()
```

In case, you want to generate your own test data and use it as an image, which is also possible. Scikit-image has `binary_blob()` functions that allows us to generate binary test data, as follows, *(figure 10.1)*

```
img = data.binary_blobs(length=512, blob_size_
fraction=0.1, seed=5)
io.imshow(img)
io.show()
```

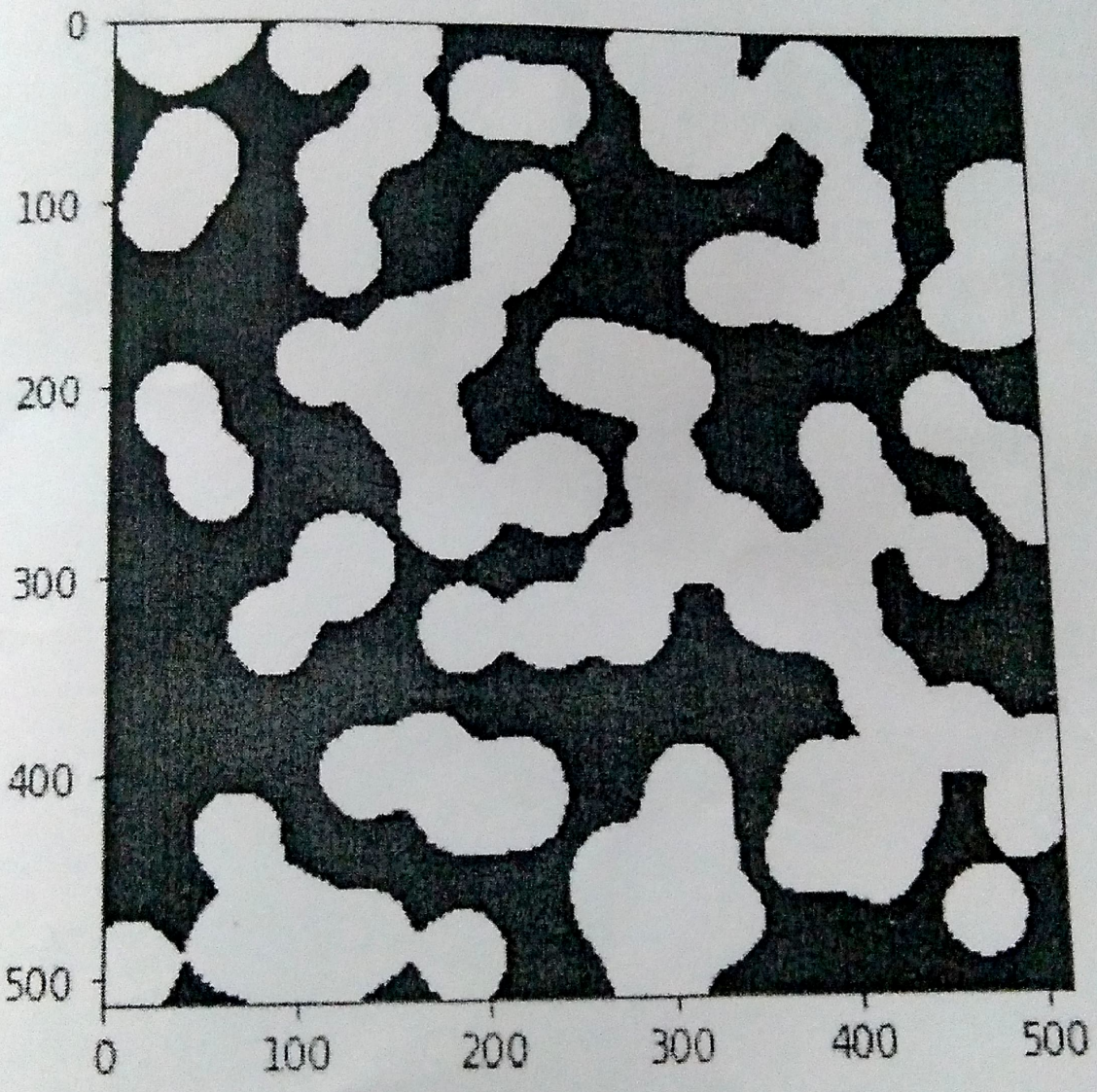

Figure 10.1 Binary blob

10.4 Colorspace Conversion

A **color model** is mathematical way of describing colors. Colorspaces map real life colors to the values in the color models. The `imread()` function

of most of the image processing libraries read the images and stores them with **RGB** as colorspace. Many times, we need to change the colorspace of an image for image processing. **Scikit-image** has functions to change the colorspace of images. Let's see the demonstration. We will use the `color` module that has all the functions to convert colorspace.

```
from skimage.color import convert_colorspace
```

Let's choose astronaut image for the demo,

```
img = data.astronaut()
plt.imshow(img)
plt.show()
```

Following code converts the image from RGB to HSV colorspace,

```
img_hsv = convert_colorspace(img, 'RGB', 'HSV')
plt.imshow(img_hsv)
plt.show()
```

We can even convert **RGB** image into greyscale using `rgb2grey()` or `rgb2gray()` functions as follows,

```
from skimage.color import rgb2gray, rgb2grey
img_gray = rgb2gray(img)
plt.imshow(img_gray, cmap = 'gray')
plt.show()
```

10.5 Summary

In this brief and short chapter, we studies how to get started with **scikit image**. We have learned how to install it on Windows and Raspberry Pi platforms. We, also, have studied the basic functions and colorspace conversion. From the next chapter onwards, we will study more advanced functionalities offered by **scikit-image**.

Exercise

Generate various test images with `binary_blob()` function. Visit https://scikit-image.org/ for more information about scikit-image library.

CHAPTER 11

Thresholding Histogram Equalization and Transformations

In the last chapter, we have learned the basics of the scikit-image. We, also, have learned how to access the images in the data module and how to create custom images for testing with `binary_blob()`.

We have, already, learned the concept of thresholding with **NumPy**. In this chapter, we will observe the same concept once again. Additionally, we will learn how to equalize image histogram to enhance the quality of images. Also, we will acquire the knowledge of how to transform images.

11.1 Simple Thresholding, Otsu's Binarization, and Adaptive Thresholding

We have already seen how to implement simple **thresholding**. In this section, we will implement the same using different style and by using `data` module from scikit-image library. Following is the code for that,

```
%matplotlib inline
import matplotlib.pyplot as plt
import skimage.data as data

img = data.camera()

thresh = 127

output1 = img > thresh
output2 = img <= thresh

output = [img, output1, output2]
```

CamScanner