CHAPTER 6

Introduction to Scientific Python

In the last chapter, you studied advanced image processing with Pillow. Pillow is a nice starting point for image processing operations. However, it has its own limitations. When it comes to implementing elaborate image processing operations like segmentation, morphological operations, advanced filters, and measurements, Pillow is inadequate. You really need to use advanced libraries for image processing. The SciPy toolkit serves as a foundation for all the scientific usage of Python. It is extensively used for a variety of scientific operations.

SciPy stands for *Scientific Python*. It extensively uses NumPy (Numerical Python) and matplotlib for numeric operations and data visualization. This chapter explains SciPy, NumPy, and matplotlib. It also explores introductory level programming examples of NumPy, scipy.misc, and matplotlib.

The Scientific Python Stack

SciPy is an open source library for scientific and technical computing in Python.

SciPy has modules for ordinary differential equations solvers, fast Fourier transforms, optimization, linear algebra, integration, interpolation, signal processing, and image processing. SciPy is used extensively by scientific, mathematical, and engineering communities around the world. There are many other libraries that use core modules of SciPy and NumPy. OpenCV and SciKit are good examples of this.

The SciPy stack has the following components:

- NumPy
- SciPy library
- matplotlib
- IPython
- Pandas
- Sympy
- Nose

A. Pajankar, *Raspberry Pi Image Processing Programming*,
DOI 10.1007/978-1-4842-2731-2_6

Figure 6-1 aptly summarizes the role of the Python SciPy stack in the world of scientific computing.

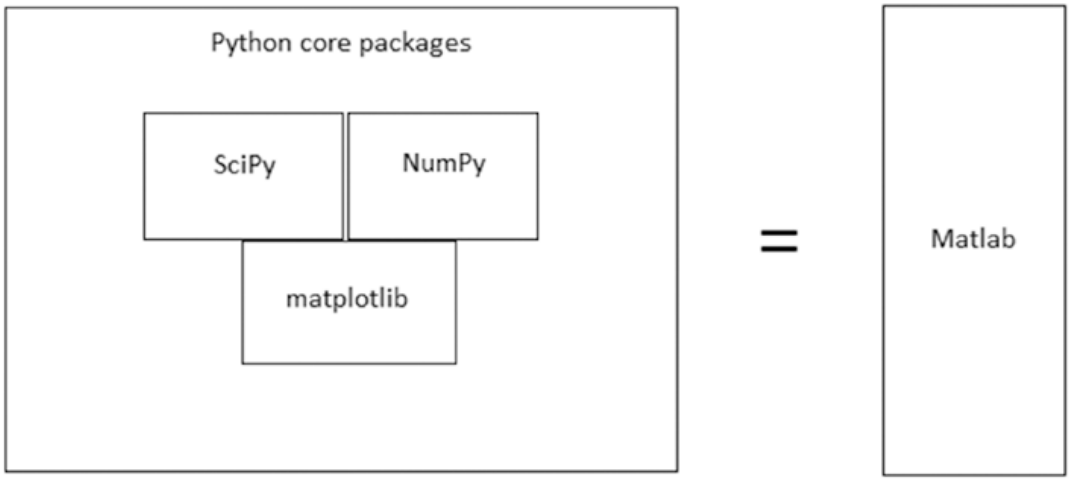

Figure 6-1. *The SciPy stack*

Installing the SciPy Stack

The best way to install the SciPy stack on Raspberry Pi is to use `pip`.

First, upgrade `pip` with the following command:

```
sudo python3 -m pip install --upgrade pip
```

Then install the SciPy stack with the following command:

```
pip3 install numpy scipy matplotlib ipython jupyter pandas sympy nose
```

This installs the entire SciPy stack.

A Simple Program

The scipy.misc module is used for basic image processing operations. Listing 6-1 shows a basic example of reading and displaying an image.

Listing 6-1. prog01.py

```
from scipy import misc

img = misc.imread('/home/pi/DIP/Dataset/4.2.01.tiff')

misc.imshow(img)
```

The code in Listing 6-1 reads an image from the path provided in the `imread()` method and then the `imshow()` method will display it using xlib.

The scipy.misc module has three built-in images, which can be used as shown in Listing 6-2.

Listing 6-2. prog02.py

```
from scipy import misc

img1 = misc.face()
img2 = misc.lena()
img3 = misc.ascent()

misc.imshow(img1)
misc.imshow(img2)
misc.imshow(img3)
```

`face()` is a face of a raccoon. `lena()` is standard test image and `ascent()` is a grayscale image.

Simple Image Processing

`scipy.misc` has three methods for simple operations. `scipy.imfilter()` applies various filters to images. Listing 6-3 shows an example.

Listing 6-3. prog03.py

```
from scipy import misc

misc.imshow(misc.imfilter(misc.face(), 'edge_enhance_more'))
```

The code in Listing 6-3, doesn't use an intermediate variable to store the image. It displays it directly by passing the `imshow()` method. The `imfilter()` method accepts two arguments. The first is the image to be filtered. The second is the type of pre-defined filter to be applied. Allowed values for the `filter-type` are `'blur'`, `'contour'`, `'detail'`, `'edge_enhance'`, `'edge_enhance_more'`, `'emboss'`, `'find_edges'`, `'smooth'`, `'smooth_more'`, and `'sharpen'`.

You can resize the image to 50 percent, as shown in Listing 6-4.

Listing 6-4. prog04.py

```
from scipy import misc

misc.imshow(misc.imresize(misc.face(), 50))
```

You can also rotate the image at a certain angle, as shown in Listing 6-5.

Listing 6-5. prog05.py

```
from scipy import misc

misc.imshow(misc.imrotate(misc.face(), 45))
```

Introduction to NumPy

Let's get started with the basics of the NumPy library in Python. Consider the code in Listing 6-6.

Listing 6-6. prog06.py

```
from scipy import misc

img = misc.face()

print(type(img))
```

The output of this program is as follows:

```
<class 'numpy.ndarray'>
```

This means that the data type of the image is `ndarray` in NumPy. In order to get started with the scientific image processing and any type of scientific programming in general, you need to know what NumPy is.

The NumPy homepage at `www.numpy.org` says this:

> *NumPy is the fundamental package for scientific computing with Python.*

It offers the following features:

- A powerful multi-dimensional array object
- Useful methods for mathematical computations
- Wrappers and tools for integration with faster C/C++ and FORTRAN code

In order to get started with the image processing with SciPy and NumPy, you need to learn basics of N-dimensional (or multi-dimensional) array objects in NumPy. Let's get started with that.

NumPy's N-dimensional array is a homogeneous (contains all the elements of the same data type) multidimensional array. It has multiple dimensions. Each dimension is known as an axis. The class corresponding to the N-dimensional array in NumPy is `numpy.ndarray`. This is what you saw in Listing 6-6. All the major image processing and computer vision libraries, like Mahotas, OpenCV, scikit-image, and scipy.ndimage (you will extensively study this last one in this book) use `numpy.ndarray` to represent images. All these libraries have `read()`, `open()`, and `imread()` methods for loading images from disk to a `numpy.ndarray` object.

NumPy and N-dimensional arrays in NumPy are such vast topics themselves that it would require volumes of books to explain them fully. Hence, you will learn the relevant and important features of these as and when they're needed. For now, you need to understand a few important `ndarray` properties that will help you understand important attributes of the images that `ndarray` represents.

Consider the code in Listing 6-7.

Listing 6-7. prog07.py

```
from scipy import misc

img = misc.face()

print(img.dtype)
print(img.shape)
print(img.ndim)
print(img.size)
```

The output is as follows:

```
uint8
(768, 1024, 3)
3
2359296
```

Let's look at what each of these means. The `dtype` attribute is for the data type of the elements that represent the image. In this case, it is `uint8`, which means an unsigned 8-bit integer. This means it can have 256 distinct values. `shape` means the dimension/size of the images. In this case, it is a color image. Its resolution is 1024x768 and it has three color channels corresponding to the colors Red, Green, and Blue. Each channel for each pixel can have one of the 256 possible values. So a combination of this can produce 256*256*256 distinct colors for each pixel.

You can visualize a color image as an arrangement of three two-dimensional planes. A grayscale image is a single plane of grayscale values. `ndim` represents the dimensions. A color image has three dimensions and a grayscale image has two dimensions. `size` represents for the total number of elements in the array. It can be calculated by multiplying the values of the dimensions. In this case, it is `768*1024*3=2359296`.

You can see the RGB value corresponding to each individual pixel, as shown in Listing 6-8.

Listing 6-8. prog08.py

```
from scipy import misc

img = misc.face()

print(img[10, 10]))
```

The code in Listing 6-8 accesses the value of the pixel located at (10, 10). The output is `[172 169 188]`.

This concludes the basics about NumPy and image processing. You will learn more about NumPy as and when needed throughout the chapters.

Matplotlib

You have used the `misc.imshow()` method for displaying an image. While this method is useful for simple applications, it is primitive. You need to use a more advanced framework for scientific applications. Matplotlib serves this purpose. It is a Matlab-style plotting and data visualization library for Python. You installed it when you installed the SciPy stack. It is an integral part of the SciPy stack. Just like NumPy, matplotlib is too a vast topic and warrants another book. The examples in this book use the pyplot module in matplotlib for the image processing requirements. Listing 6-9 shows a simple program for the image processing.

Listing 6-9. prog09.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.face()

plt.imshow(img)
plt.show()
```

The code in Listing 6-9 imports the pyplot module. The `imshow()` method adds the image to the plot window. The `show()` method shows the plot window. The output is shown in Figure 6-2.

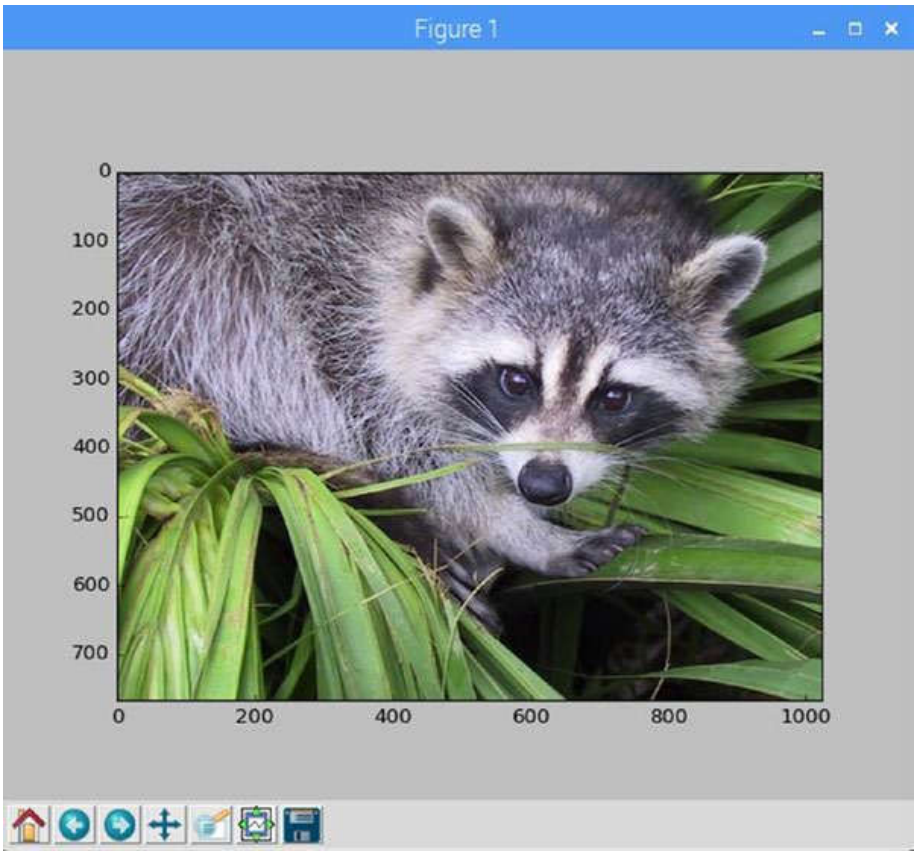

Figure 6-2. *The pyplot demo*

You can also turn off the axes (or the ruler) and add a title to the image, as shown in Listing 6-10.

Listing 6-10. prog10.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.lena()

plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('face')
plt.show()
```

As the image is grayscale, you have to choose a `gray` color map in the `imshow()` method so the image color space is properly displayed in the plot window. `axis('off')` is used to turn the axes off. The `title()` method is used to specify the title of the image. The output is shown in Figure 6-3.

Figure 6-3. *Lena image with title and axes off*

You can use `imshow()` to push multiple images to an image grid in the plot window, as shown in Listing 6-11.

Listing 6-11. prog11.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img1 = misc.face()
img2 = misc.ascent()
img3 = misc.lena()

titles = ['face', 'ascent', 'lena']
images = [img1, img2, img3]

plt.subplot(1, 3, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(1, 3, 2)
```

```
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(1, 3, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.show()
```

You have used the `subplot()` method before, with `imshow()`. The first two arguments in the `subplot()` method specify the dimensions of the grid and the third argument specifies the position of the image in the grid. The numbering of the images in the grid starts from the top-left edge. The top-left position is the first position, the next position is the second one, and so on. The result is shown in Figure 6-4.

Figure 6-4. *Multiple image grid*

Image Channels

You can separate image channels of a multi-channel image. The code for that process is shown in Listing 6-12.

Listing 6-12. Prog12.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

titles = ['face', 'Red', 'Green', 'Blue']
images = [img, r, g, b]
```

```
plt.subplot(2, 2, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(2, 2, 2)
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(2, 2, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.subplot(2, 2, 4)
plt.imshow(images[3], cmap='gray')
plt.axis('off')
plt.title(titles[3])

plt.show()
```

The result of the code in Listing 6-12 is shown in Figure 6-5.

Figure 6-5. *Separate image channels*

You can use the `np.dstack()` method, which merges all the channels, to create the original image, as shown in Listing 6-13.

Listing 6-13. prog13.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt
import numpy as np

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

output = np.dstack((r, g, b))

plt.imshow(output)
plt.axis('off')
plt.title('Combined')
plt.show()
```

Run the code in Listing 6-13 to see the workings of the `np.dstack()` for yourself.

Conversion Between PIL Image Objects and NumPy ndarrays

You can use the `np.asarray()` and `Image.fromarray()` methods to convert between PIL images and NumPyndarrays, as shown in Listing 6-14.

Listing 6-14. prog14.py

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open('/home/pi/DIP/Dataset/4.2.01.tiff')

print(type(img))
img.show()

num_img = np.asarray(img)
plt.imshow(num_img)
plt.show()
print(type(num_img))
```

```
img = Image.fromarray(np.uint8(num_img))

print(type(img))
img.show()
```

The console output is as follows:

```
<class 'PIL.TiffImagePlugin.TiffImageFile'>
<class 'numpy.ndarray'>
<class 'PIL.Image.Image'>
```

You can use the `misc.fromimage()` and `misc.toimage()` methods to achieve the same conversion, as shown in Listing 6-15.

Listing 6-15. prog15.py

```
from PIL import Image
import scipy.misc as misc
import matplotlib.pyplot as plt

img = Image.open('/home/pi/DIP/Dataset/4.2.01.tiff')

print(type(img))
img.show()

num_img = misc.fromimage(img)

print(type(num_img))
plt.imshow(num_img)
plt.show()

img = misc.toimage(num_img)

print(type(img))
img.show()
```

The console output of Listing 6-15 is the same as the earlier example shown in Listing 6-14.

Conclusion

In this chapter, you were introduced to SciPy stack, NumPy, and matplotlib. You also explored the scipy.misc module for basic image processing and conversion. In the next chapter, you will start exploring the scipy.ndimage module for more image processing operations.

CHAPTER 7

Transformations and Measurements

In the last chapter, you were introduced to the scientific Python stack. You learned the basics of NumPy and matplotlib. You learned about the useful modules called ndarray and pyplot from NumPy and matplotlib respectively. You also learned about the `scipy.misc` module and basic image processing with it. In this chapter, you will further explore SciPy. You will learn to use the `scipy.ndimage` library for processing images. You will explore the methods for transformations and measurements. This is a short and simple chapter.

Transformations

You studied a few basic transformations with `scipy.misc` in the last chapter. Here, you will look at few more.

Note Just like with `scipy.misc`, `scipy.ndimage` has an `imread()` method that serves the same purpose as `scipy.misc.imread()`. You will be using `face()`, `lena()`, and `ascent()` throughout the rest of the book. However, if you want to use other images in the `Dataset` directory, you can use `imread()` from the `misc` or `ndimage` module in SciPy.

Let's get started with the simple transformation of shifting. The `shift()` method accepts the image and the values to be applied to the coordinates for shifting as arguments. The example is shown in Listing 7-1.

Listing 7-1. prog01.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt
```


A. Pajankar, *Raspberry Pi Image Processing Programming*,
DOI 10.1007/978-1-4842-2731-2_7

```
img = misc.lena()

output = ndi.shift(img, [20, -20])

plt.imshow(output, cmap='gray')
plt.title('Shift Demo')
plt.axis('off')
plt.show()
```

Figure 7-1 shows the output.

Figure 7-1. *Shift demo*

You can also zoom in on the image using the `zoom()` method. You have to pass the image and the scale of zooming for each of the axes as arguments to the method. Listing 7-2 shows an example of doing this.

Listing 7-2. prog02.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = misc.lena()
```

```
plt.imshow(ndi.zoom(img, [5, 3]), cmap='gray')
plt.title('Zoom Demo')
plt.axis('off')

plt.show()
```

The output of Listing 7-2 is shown in Figure 7-2.

Figure 7-2. *Zoom demo*

Measurements

In Chapter 5, you learned how to create a histogram. SciPy also has a `histogram()` method that computes a histogram of image channels. You can use matplotlib to display the histogram. Listing 7-3 shows an example of a histogram computation with SciPy; it's been plotted using matplotlib.

Listing 7-3. prog03.py

```
import scipy.misc as misc
import scipy.ndimage as ndi
import matplotlib.pyplot as plt

img = misc.lena()
```

```
hist = ndi.histogram(img, 0, 255, 256)

plt.plot(hist, 'k')
plt.title('Lena Histogram')
plt.grid(True)

plt.show()
```

This program passes the image, the minimum pixel value, the maximum pixel value, and the number of bins as arguments to the `histogram()` method. It passes the histogram array returned by the `histogram()` method and the color of the histogram graph (`'k'`) as arguments to the `plot()` method. The resultant histogram is shown in Figure 7-3.

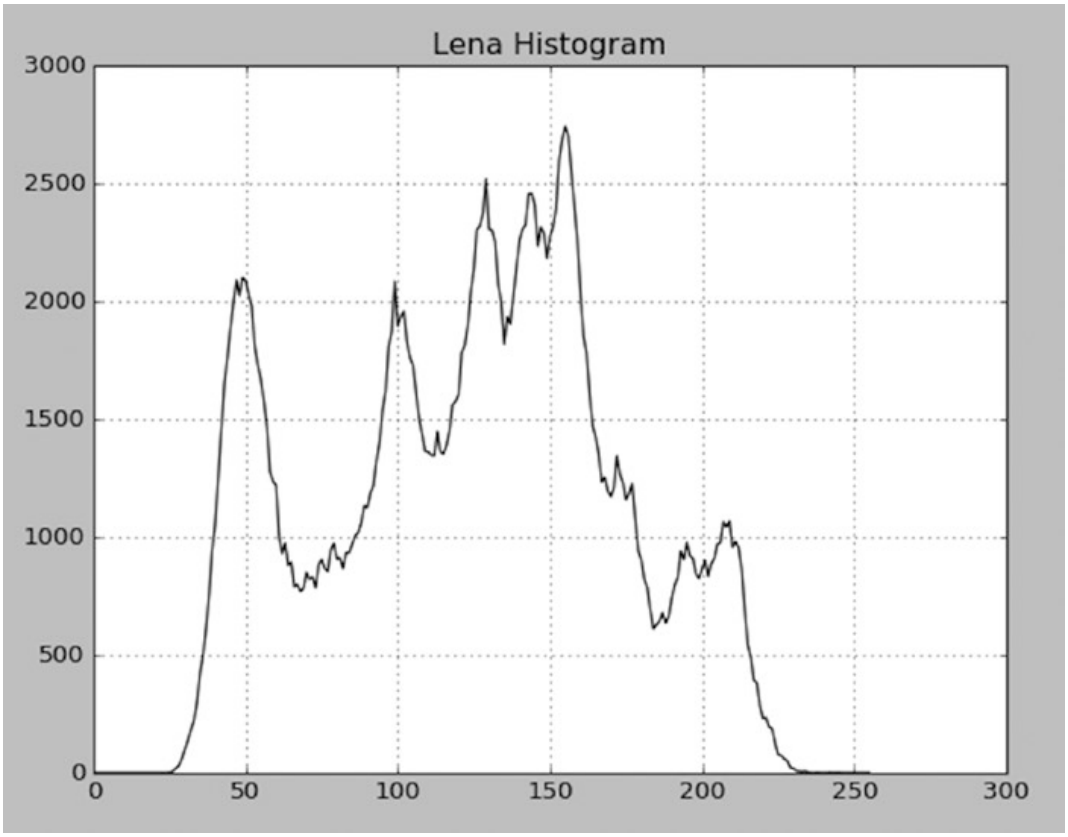

Figure 7-3. *Histogram of the Lena image*

The `minimum()` and `maximum()` methods return the minimum and maximum values of the pixels, respectively. The `minimum_position()` and `maximum_position()`methods return the values of the positions of pixels with the minimum and maximum intensities, respectively. The `extrema()` method combines the functionalities of the four methods. The demonstration is shown in Listing 7-4.

Listing 7-4. prog04.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.minimum(img))
print(ndi.minimum_position(img))
print(ndi.maximum(img))
print(ndi.maximum_position(img))

print(ndi.extrema(img))
```

The output is as follows:

```
0
(201, 268)
255
(190, 265)
(0, 255, (201, 268), (190, 265))
```

The first line is the value of the lowest intensity pixel. The second line is its position. The next two lines refer to the value of the highest intensity pixel and its position. The next line is the output of the `extrema()` method, which combines all the previous output.

The program shown in Listing 7-5 demonstrates the statistical information about the image pixels. The methods work on the intensity values of the pixel; the names of methods are fairly self-explanatory.

Listing 7-5. prog05.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.sum(img))
print(ndi.mean(img))
print(ndi.median(img))
print(ndi.variance(img))
print(ndi.standard_deviation(img))
```

The output is as follows:

```
22932324
87.4798736572
80.0
2378.9479363
48.7744598771
```

You can also calculate the center of mass based on the intensity values of the pixels, as shown in Listing 7-6.

Listing 7-6. prog06.py

```
import scipy.misc as misc
import scipy.ndimage as ndi

img = misc.ascent()

print(ndi.center_of_mass(img))
```

This will produce the following output:

```
(259.99734235396289, 254.60907272197969)
```

EXERCISE

Complete the following exercises to gain an in-depth understanding of transformations and measurements.

- Explore the `ndimage.rotate()` method. Write the code for using it.
- We calculated the histogram for a grayscale image. Calculate the histogram for a color image by computing it separately for each channel. Use the relevant color for the histogram graph for displaying using pyplot.
- We calculated the measurements for grayscale images. Do the same for color images.

Conclusion

In this chapter, you explored the methods for transformations and measurements. You studied the shift and zoom transformations. You calculated the histogram of a grayscale image. You also calculated statistical information about the images.

In the next chapter, you will study the image kernels and filters, their types, and their applications in the image enhancement in detail.