Chapter 3

Image and Its Properties

3.1 Introduction

We begin this chapter with an introduction to images, image types, and data structures in Python. Image processing operations can be imagined as a workflow similar to Figure 3.1. The workflow begins with reading an image. The image is then processed using either low-level or high-level operations. Low-level operations operate on individual pixels. Such operations include filtering, morphology, thresholding, etc. High-level operations include image understanding, pattern recognition, etc. Once processed, the image(s) are either written to disk or visualized. The visualization may be performed during the course of processing as well. We will discuss this workflow and the functions using Python as an example.

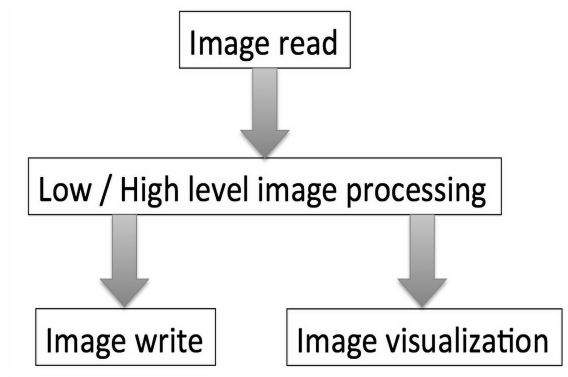


FIGURE 3.1: Image processing work flow.

3.2 Image and Its Properties

In the field of medical imaging, the images may span all spatial dimensions (x-, y- and x-axis) and also the time dimension. Hence it is common to find images in 3D, and in some cases such as cardiac CT, images in 4D. In the case of optical microscopy, the images of the same specimen may be acquired at various emission and excitation wavelengths. Such images will span multiple channels and may have more than 4 dimensions. We begin the discussion by clarifying some of the mathematical terms that are used in this book.

For simplicity, let us assume the images that will be discussed in this book are 3D volumes. A 3D volume (I) can be represented mathematically as

$$\alpha = I \longrightarrow \mathbb{R} \text{ and } I \subset \mathbb{R}$$

Thus, every pixel in the image has a real number as its value. However, in reality as it is easier to store integers than to store floats; most images have integers for pixel values.

3.2.1 Bit-Depth

The pixel range of a given image format is determined by its bit-depth. The range is $[0, 2^{bitdepth} - 1]$. For example, an 8-bit image will have a range of $[0, 2^8 - 1] = [0, 255]$. An image with higher bit-depth needs more storage in disk and memory. Most of the common photographic formats such as JPEG, PNG, etc. use 8 bits for storage and only have positive values.

Medical and microscope images use a higher bit-depth, as scientific applications demand higher accuracy. A 16-bit medical image will have values in the range $[0, 65535]$ for a total number 65536 $(= 2^{16})$ values. For a 16-bit image that has both positive and negative pixel values, the range is $[-32768, +32767]$. The total number of values in this case is

65536 ($= 2^{16}$) or a bit-depth of 16. A good example of such an image is a CT DICOM image.

Scientific image formats store the pixel values at high precision, not only for accuracy, but also to ensure that physical phenomena records are not lost. In CT, for example, a pixel value of > 1000 indicates bone. If the image is stored in 8-bit, the pixel value of bone would be truncated at 255 and hence the information will be permanently lost. In fact, the most significant pixels in CT have intensity > 255 and hence need higher bit-depth.

There are a few image formats that store images at even higher bit-depth such as 32 or 64. For example, a JPEG image containing RGB (3 channels) will have a bit-depth of 8 for each channel and hence has a total bit-depth of 24. Similarly, a TIFF microscope image with 5 channels (say) with each channel at 16-bit depth will have a total bit-depth of 80.

3.2.2 Pixel and Voxel

A pixel in an image can be thought of as a bucket that collects light or electrons depending on the type of detector used. A single pixel in an image spans a distance in the physical world. For example, in Figure 3.2, the arrows indicate the width and height of a pixel placed adjacent to three other pixels. In this case, the width and height of this pixel is 0.5 mm. Thus in a physical space, traversing a distance of 0.5 mm is equivalent to traversing 1 pixel in the pixel space. For all practical purposes, we can assume that detectors have square pixels, i.e., the pixel width and pixel height are the same.

The pixel size could be different for different imaging modalities and different detectors. For example, the pixel size is greater for CT compared to micro-CT.

In medical and microscope imaging, it is more common to acquire 3D images. In such cases, the pixel size will have a third dimension,

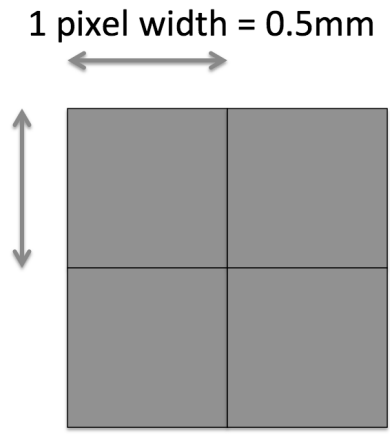
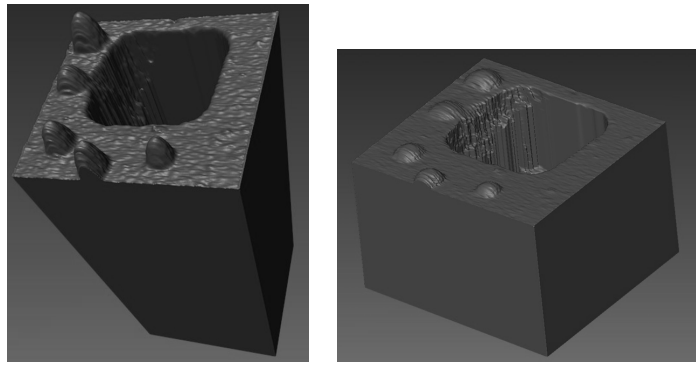


FIGURE 3.2: Width and height of pixel in physical space.

namely the pixel depth. The term pixel is generally applied to 2D and is replaced by voxel in 3D images.

Most of the common image formats like DICOM, nifti, and some microscope image formats contain the voxel size in their header. Hence, when such images are read in a visualization or image processing program, an accurate analysis and visualization can be performed. But if the image does not have the information in the header or if the visualization or image processing program cannot read the header properly, it is important to use the correct voxel size for analysis.

Figure 3.3 illustrates the problem of using an incorrect voxel size in visualization. The left image is the volume rendering of an optical coherence tomography image with incorrect voxel size in the z-direction. The right image is the volume rendering of the same image with correct voxel size. In the left image, it can be seen clearly that the object is highly elongated in the z-direction. In addition, the undulations at the top of the volume and the five hilly structures at the top are also made prominent by the incorrect voxel size. The right image has the same shape and size as the original object. The problem not only affects visualization but also any measurements performed on the volume.



(a) Volume rendering with incorrect voxel size. The 3D is elongated in the z-direction.

(b) Volume rendering with correct voxel size.

FIGURE 3.3: An example of volume rendering with correct and incorrect voxel size.

3.2.3 Image Histogram

A histogram is a graphical depiction of the distribution of pixel value in an image. The image in Figure 3.4 is a histogram of an image. The x-axis is the pixel value and the y-axis is the frequency or the number of pixels with the given pixel value. In the case of an integer-based image such as JPEG, whose values span $[0, 255]$, the number of values in the x-axis will be 256. Each of these 256 values is referred to as a "bin." Several bins can also be used in the x-axis. In the case of images containing floating-point values, the bins will have a range of values.

Histograms are a useful tool in determining the quality of the image. A few observations can be made in Figure 3.4:

1. The left side of the histogram corresponds to lower pixel values. Hence if the frequency at lower pixel values is very high, it indicates that some of the pixels might be missing from that end,

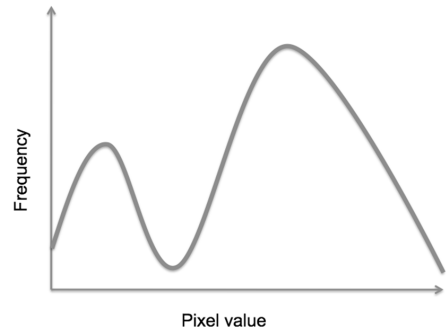


FIGURE 3.4: An example of a histogram.

i.e., there are values farther left of the first pixel that were not recorded in the image.

2. An ideal histogram should have close to zero frequency for the lower pixel values.

3. The right side of the histogram corresponds to higher pixel values. Hence, if the frequency at higher pixel values is very high, it indicates saturation, i.e., there might be some pixels to the right of the highest value that were never recorded.

4. An ideal histogram should have close to zero frequency for the higher pixel values.

5. The above histogram is bi-modal. The trough between the two peaks is the pixel value that can be used for segmentation by thresholding. But not all images have bi-modal histograms; hence there are many techniques for segmentation using histograms. We will discuss some of these techniques in Chapter 8, "Segmentation."

3.2.4 Window and Level

The human eye can view a large range of intensity values, while modern displays are severely limited in their capabilities.

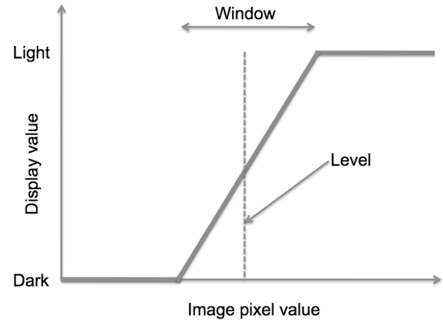


FIGURE 3.5: Window and level.

Image viewing applications display the pixel value after a suitable transformation due to the fact that displays have a lower intensity range than the intensity range in an image. One example of the transformation, namely window-level, is shown in Figure 3.5. Although the computer selects a transformation, the user can modify it by changing the window range and the level. The window allows modifying the contrast of the display while the level changes the brightness of the display.

3.2.5 Connectivity: 4 or 8 Pixels

The usefulness of this section will be more apparent with the discussion of convolution in Chapter 7, "Fourier Transform." During the convolution operation, a mask or kernel is placed on top of an image pixel. The final value of the output image pixel is determined using a linear combination of the value in the mask and the pixel value in the image. The linear combination can be calculated for either 4-connected pixels or 8-connected pixels. In the case of 4-connected pixels shown in Figure 3.6, the process is performed on the top, bottom, left and right pixels. In the case of 8-connected pixels, the process is performed in addition on the top-left, top-right, bottom-left and bottom-right pixels.

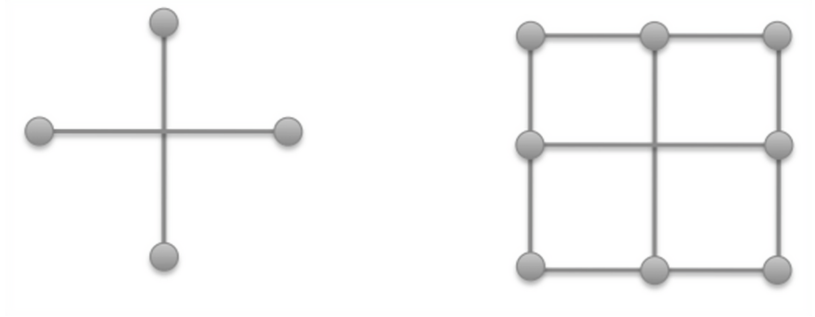


FIGURE 3.6: An example of 4 and 8 pixel connectivity.

3.3 Image Types

There are more than 100 image formats. Some of these formats, such as JPEG, GIF, PNG, etc., are used for photographic images. Formats such as DICOM, NIFTI, and Analyze AVW are used in medical imaging. Formats such as TIFF, ICS, IMS, etc., are used in microscope imaging. In the following sections, we discuss some of these formats.

3.3.1 JPEG

JPEG stands for the Joint Photographic Experts Group, a joint committee formed to add images to text terminals. Its extension is .jpg or .jpeg. It is one of the most popular formats due to its ability to compress the data significantly with minimal visual loss. In the initial days of the World Wide Web, JPEG became popular as it helped save bandwidth in image data transfer. It is a lossy format, that compresses data using Discrete Cosine Transform (DCT). The parameters of compression can be tuned to minimize the loss in detail. Since JPEG stores image data after transforming them using DCT, it is not very suitable for storing images that contain fine structures such as lines, curves, etc. Such images are better stored as PNG or TIFF. The JPEG images can be viewed using viewers that are built into most computers. Since JPEG images can be compressed, image standards such as TIFF

and DICOM may use JPEG compression to store the image data when compression is needed.

3.3.2 TIFF

TIFF stands for Tagged Image File Format. Its extension is .tif or .tiff. The latest version of the TIFF standards is 6.0. It was created in the 80's for storing and encoding scanned documents. It was developed by Aldus Corporation, which was later acquired by Adobe Systems. Hence, the copyright for TIFF standards is held by Adobe Systems.

Originally it was developed for single-bit data but today's standards allow storage of 16-bit and even floating-point data. Charged Couple Device (CCD) cameras used in scientific experiments acquire images at more than 12-bit resolution and hence TIFF images that store high precision are used extensively. The TIFF images can be stored internally using JPEG lossy compression or can be stored with lossless compression such as LZW.

It is popular in the microscope community for the fact that it has higher bit-depth (> 12 bits) per pixel per channel and also for its ability to store a sequence of images in a single TIFF file. The latter is sometimes referred as 3D TIFF. Most of the popular image processing software for the microscope community can read most forms of TIFF images. Simple TIFF images can be viewed using viewers built into most computers. The TIFF images generated from scientific experiments are best viewed using applications that are specialized in that domain.

3.3.3 DICOM

Digital Imaging and Communication in Medicine (DICOM) is a standard format for encoding and transmitting medical CT and MRI data. This format stores the image information along with other data like patient details, acquisition parameters etc. DICOM images are used by doctors in various disciplines such as radiology, neurology, surgery, cardiology, oncology, etc. There are more than 20 DICOM committees

that meet and update the standards 4 or 5 times a year. It is managed by the National Electrical Manufacturers Association (NEMA), which owns the copyright of the DICOM standards.

DICOM format uses tested tools such as JPEG, MPEG, TCP/IP for its internal working. This allows easier deployment and creation of DICOM tools. DICOM standards also define the transfer of images, storage, and other allied workflows. Since DICOM standards have become popular, many image processing readers and viewers have been created to read, process and write images.

DICOM images have header as well as image data similar to other image formats. But, unlike other format headers, the DICOM header contains not only information about the size of the image, pixel size, etc., but also patient information, physician information, imaging parameters, etc. The image data may be compressed using various techniques like JPEG, lossless JPEG, run length encoding (RLE), etc. Unlike other formats, DICOM standards define both the data format and also the protocol for transfer.

The listing below is a partial example of a DICOM header. The patient and doctor information have been either removed or altered for privacy. Section 0010 contains patient information, section 0009 details the CT machine used for acquiring the image, and section 0018 details the parameter of acquisition, etc.

```
0008,0022  Acquisition Date: 20120325
0008,0023  Image Date: 20120325
0008,0030  Study Time: 130046
0008,0031  Series Time: 130046
0008,0032  Acquisition Time: 130105
0008,0033  Image Time: 130108
0008,0050  Accession Number:
0008,0060  Modality: CT
0008,0070  Manufacturer: GE MEDICAL SYSTEMS
0008,0080  Institution Name: ----------------------
```

```
0008,0090  Referring Physician's Name: XXXXXXX
0008,1010  Station Name: CT01_OC0
0008,1030  Study Description: TEMP BONE/ST NECK W
0008,103E  Series Description: SCOUTS
0008,1060  Name of Physician(s) Reading Study:
0008,1070  Operator's Name: ABCDEF
0008,1090  Manufacturer's Model Name: LightSpeed16
0009,0010  ---: GEMS_IDEN_01
0009,1001  ---: CT_LIGHTSPEED
0009,1002  ---: CT01
0009,1004  ---: LightSpeed16
0010,0010  Patient's Name: XYXYXYXYXYXYX
0010,0020  Patient ID: 213831
0010,0030  Patient's Birth Date: 19650224
0010,0040  Patient's Sex: F
0010,1010  Patient's Age:
0010,21B0  Additional Patient History:
           ? MASS RIGHT EUSTACHIAN TUBE
0018,0022  Scan Options: SCOUT MODE
0018,0050  Slice Thickness: 270.181824
0018,0060  kVp: 120
0018,0090  Data Collection Diameter: 500.000000
0018,1020  Software Versions(s): LightSpeedverrel
0018,1030  Protocol Name: 3.2 SOFT TISSUE NECK
0018,1100  Reconstruction Diameter:
0018,1110  Distance Source to Detector: 949.075012
0018,1111  Distance Source to Patient: 541.000000
0018,1120  Gantry/Detector Tilt: 0.000000
0018,1130  Table Height: 157.153000
0018,1140  Rotation Direction: CW
0018,1150  Exposure Time: 2772
0018,1151  X-ray Tube Current: 10
0018,1152  Exposure: 27
```

```
0018,1160  Filter Type: BODY FILTER
0018,1170  Generator Power: 1200
0018,1190  Focal Spot(s): 0.700000
0018,1210  Convolution Kernel: STANDARD
```

The various software that can be used to manipulate DICOM images can be found online. We will classify these software based on the user requirements.

The user might need:

1. A simple viewer with limited manipulation like ezDICOM [Ror20].

2. A viewer with ability to manipulate images and perform rendering like Osirix [SAR20].

3. A viewer with image manipulation capability and also extensible with plugins like ImageJ.

1. **ezDICOM:** This is a viewer that provides sufficient functionality that allows users to view and save DICOM files without installing any other complex software in their system. It is available only for Windows OS. It can read DICOM files and save them in other file formats. It can also convert image files to analyze format.

2. **Osirix:** This is a viewer with extensive functionality and is available free, but unfortunately it is available only in MacOSX. Like other DICOM viewers, it can read and store files in different file formats and as movies. It can perform Multi-Planar Reconstruction (MPR), 3D surface rendering, 3D volume rendering, and endoscopy. It can also view 4D DICOM data. The surface rendered data can also be stored as VRML, STL files, etc.

3. **ImageJ:** ImageJ was funded by the National Institutes of Health (NIH) and is available as open source. It is written in Java and users can add their own Java classes or plugins. It is available

in all major operating system like Windows, Linux, UNIX, Mac, etc. It can read all DICOM formats and can store the data in various common file formats and also as movies. The plugins allow various image processing operations. Since the plugins can be easily added, the complexity of the image processing operation is limited only by the user's knowledge of Java. Since ImageJ is a popular image processing software, a brief introduction is presented in Appendix C, "Introduction to ImageJ."

3.4 Data Structures for Image Analysis

Image data is generally stored as a mathematical matrix. So in general, a 2D image of size 1024-by-1024 is stored in a matrix of the same size. Similarly, a 3D image is stored in a 3D matrix. In numpy, a mathematical matrix is called a numpy array. As we will be discussing in the subsequent chapters, the images are read and stored as a numpy array and then processed using either functions in a Python module or user-defined functions.

Since Python is a dynamically typed language (i.e., no defining data type), it will determine the data type and size of the image at run time and store appropriately.

3.5 Reading, Writing and Displaying Images

3.5.1 Reading Images

After a lot of research, we decided to use Python's computer vision module, OpenCV [Ope20a] for reading and writing images, the PIL module's Image for reading images, and Matplotlib's pyplot to display images.

OpenCV is imported as cv2. We use imread function to read an image, which returns an ndarray. The cv2.imread supports the following file formats:

- Windows bitmaps: bmp, dib
- JPEG files: jpeg, jpg, jpe
- JPEG 2000 files: jp2
- Portable Network Graphics: png
- Portable image format: pbm, pgm, ppm
- TIFF files: tiff, tif

Below is the cv2's code snippet for reading images.

```
import cv2

# Reading image and converting it into an ndarray
img = cv2.imread('Picture1.png')

# Converting img to grayscale
img_grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

We import cv2 and use the cv2.imread function to read the image as an ndarray. To convert a color image to grayscale, we use the function cvtColor whose first argument is the ndarray of an image and the second argument is cv2.COLOR_BGR2GRAY. This argument, cv2.COLOR_BGR2GRAY, converts an RGB image, which is a three-channel ndarray to grayscale, which is a single-channel ndarray using the following formula:

$$y = 0.299 * R + 0.587 * G + 0.114 * B \tag{3.1}$$

Another way to read images would be using PIL module's Image class. Below is the code snippet for reading images using PIL's Image.

```
from PIL import Image
import numpy as np

# Reading image and converting it into grayscale.
img = Image.open('Picture2.png').convert('L')
# convert PIL Image object to numpy array
img = np.array(img)

# Performing image processing on img.
img2 = image_processing(img)

# Converting ndarray to image for saving using PIL.
im3 = Image.fromarray(img2)
```

In the above code, we import Image from the PIL module. We open the 'Picture.png' image and convert a three-channel image to a single-channel grayscale by using convert('L') and the result is a PIL Image object. We then convert this PIL Image object to a numpy ndarray using the np.array function because most image processing modules in Python can only handle a numpy array and not a PIL Image object. After performing some image processing operation on this ndarray, we convert the ndarray back to an image using Image.fromarray, so that it can be saved or visualized.

3.5.2 Reading DICOM Images using pyDICOM

We will use pyDICOM [Mas20], a module in Python to read or write or manipulate DICOM images. The process for reading DICOM images is similar to JPEG, PNG, etc. Instead of using cv2, the pyDICOM module is used. The pyDICOM module is not installed by default in the distributions. Please refer to the pyDICOM documentation at [Mas20]. To read a DICOM file, the DICOM module is first imported. The file is then read using the "read_file" function.

```
import dicom
ds = dicom.read_file("ct_abdomen.dcm")
```

3.5.3 Writing Images

Throughout this book, to write or save images we will use cv2.imwrite. The cv2.imwrite function supports the following file formats:

- JPEG files: *.jpeg, *.jpg, *.jpe
- Portable Network Graphics: *.png
- Portable image format: *.pbm, *.pgm, *.ppm
- TIFF files: *.tiff, *.tif

Here is an example code snippet where we read an image and write an image. The imwrite function takes the file name and the ndarray of an image as input. The file format is identified using the file extension in the file name.

```
import cv2
img = cv2.imread('image1.png')

# cv2.imwrite will take an ndarray.
cv2.write('file_name', img)
```

In the subsequent chapters, we will continue to use the above approach for writing or saving images.

3.5.4 Writing DICOM Images using pyDICOM

To write a DICOM file, the DICOM module is first imported. The file is then written using the "write_file" function. The input to the function is the name of the DICOM file and also the array that needs to be stored.

```
import dicom
datatowrite = ...
dicom.write_file("ct_abdomen.dcm",datatowrite)
```

3.5.5 Displaying Images

Throughout this book, to display images, we will use Matplotlib.pyplot. Below is a sample code snippet that reads and displays an image.

```
import cv2
import matplotlib.pyplot as plt

# cv2.imread will read the image and convert it into an
ndarray.
img = cv2.imread('image1.png')

# We import matplotlib.pyplot to display an image in
grayscale.
# If gray is not supplied the image will be displayed
in color.
plt.imshow(img, 'gray')
plt.show()
```

We are importing cv2 and matplotlib.pyplot modules. Notice that we are aliasing matplotlib.pyplot as plt. We use cv2.imread to read an image and we use plt.imshow to display the image. As we want a grayscale image to be displayed, we provide a string 'gray' to the plt.imshow function.

Note: We can also display a DICOM image using plt.imshow because pyDICOM's read_file also returns a data object that contains the image data as an ndarray.

3.6 Programming Paradigm

As described in the introductory section, the workflow (Figure 3.1) for image processing begins with reading an image and finally ends with

either writing the image to file or visualizing it. The image processing operations are performed between the reading and writing or visualizing the image. In this section, the code snippet that will be used for reading and writing or visualizing the image is presented. This code snippet will be used in most of the programs presented in this book.

Below is a sample code where cv2 and matplotlib are used.

```
# cv2 module's imread to read an image as an ndarray.
# cv2 module's imwrite to write an image.
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('image1.png')

# Converting img to grayscale (if needed).
img_grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# We process img_grayscale and obtain img_processed.
# The function image_processing can perform any image
# processing or computer vision operation
img_processed = image_processing(img_grayscale)

# cv2.imwrite will take an ndarray and store it.
cv2.write('file_name.png', img_processed)

# We import matplotlib.pyplot to display an image in
grayscale.
plt.imshow(img_processed, 'gray')
plt.show()
```

In the above code, the cv2 module is imported. Then matplotlib .pyplot is imported as plt. We use cv2.imread to read image1.png and return an ndarray. We use cv2.cvtColor along with the argument cv2.COLOR_BGR2GRAY to convert img, which is a three-channel ndarray to a single-channel ndarray and we store it in img_grayscale.

We perform some image processing (assuming that the function image_processing already exists) on img_grayscale and we assign it to img_processed. We save img_processed using cv2.write, which converts the ndarray to an image. To visualize, we use plt.imshow. The plt.imshow function takes the ndarray as a necessary input argument and image type as an optional argument. In this case, for image type, we chose gray.

Below is another sample code where PIL and matplotlib are used instead of cv2 for reading and writing images.

```
# PIL module to read and save an image.
from PIL import Image
import matplotlib.pyplot as plt

# Opening image and converting it into grayscale.
img = Image.open('image2.png').convert('L')
# convert PIL Image object to numpy array
img = np.array(img)

# We process img_grayscale and obtain img_processed
img_processed = image_processing(img)

# Converting ndarray to a PIL Image.
img_out = Image.fromarray(img_processed)

# Save the image to a file.
img_out.save('file_name.png')

# Display the image in grayscale
plt.imshow(img_processed, 'gray')
plt.show()
```

In the above code, Image class is imported from PIL. Then matplotlib is imported as plt. We use Image.open to read the image and in the same line, using convert('L'), the image is converted from a three-channel

image to a single-channel Image object. We use the np.array function to convert the PIL image to ndarray. We then perform some image processing (assuming that the function image_processing already exists) on img and we assign it to img_processed. We use Image.fromarrray to convert img_processed, which is an ndarray, to a PIL Image object. We save img_processed using the save method in the PIL Image class. To visualize, we use plt.imshow. The plt.imshow function takes the ndarray as a necessary input argument and image type as an optional argument. In this case, for image type, we chose gray.

3.7 Summary

- Image processing is preceded by reading an image file. It is then followed by either writing the image to file or visualization.
- An image is stored generally in the form of matrices. In Python, it is processed as a numpy n-dimensional array or ndarray.
- An image has various properties like bit-depth, pixel/voxel size, histogram, window-level, etc. These properties affect the visualization and processing of images.
- There are hundreds of image formats created to serve the needs of the image processing community. Some of these formats like JPEG, PNG, etc. are used generally for photographs while DICOM, Analyze AVW, and NIFTI are used for medical image processing.
- In addition to processing these images, it is important to view these images using graphical tools such as ezDicom, Osirix, ImageJ, etc.
- Reading and writing images can be performed using many methods. One such method was presented in this chapter. We will continue to use this method in all the subsequent chapters.

3.8 Exercises

1. An image of size 100-by-100 has isotropic pixel size of 2-by-2 microns. The number of pixels in the foreground is 1000. What is the area of the foreground and background in $microns^2$?

2. A series of images are used to create a volume of data. There are 100 images each of size 100-by-100. The voxel size is 2-by-2-by-2 microns. Determine the volume of the foreground in $microns^3$ given the number of pixels in the foreground is 10,000.

3. A histogram plots the frequency of occurrence of the various pixel values. This plot can be converted to a probability density function or pdf, so that the y-axis is the probability of the various pixel values. How can this be accomplished?

4. To visualize window or level, open an image in any image processing software (such as ImageJ). Adjust window and level. Comment on the details that can be seen for different values of window and level.

5. There are specialized formats for microscope images. Conduct research on these formats.