**Introduction**

Before diving straight into image processing let us understand images first. An image as humans see is a two dimensional grid with each cell in the grid filled with a color value. Each cell of the grid is formally called a pixel (abbreviation of picture element). A computer also sees the image in the same way. An image on a computer is a two dimensional matrix of numbers with each cell in the matrix storing the corresponding pixel color value in the image.

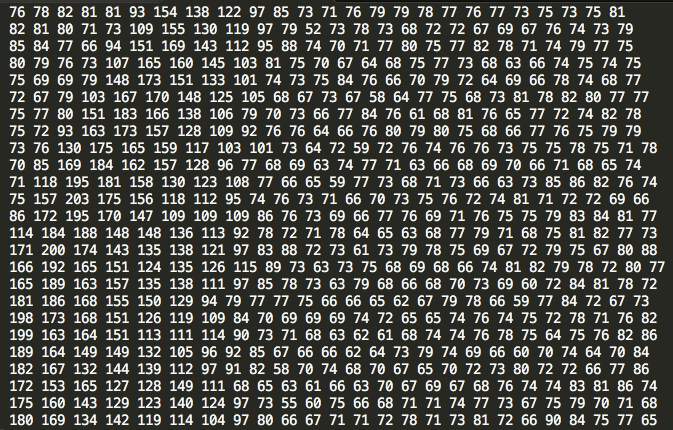
Following illustration will make it more clear.



Figure 1 This figure shows the image matrix as stored on a computer of a small portion of the image in the red box.

Let us take a look at some common applications of image processing:

* Medicine

In the recent years, the field of Medicine has seen rapid advancements. For example, more sophisticated imaging techniques, better techniques to detect the nature of tumors in MRI/PET scans. The interdisciplinary research between biology and image processing played an important role. The following image illustrates how image processing algorithms are being used to detect tumors. This has helped in early diagnosis of diseases and a more effective treatment.

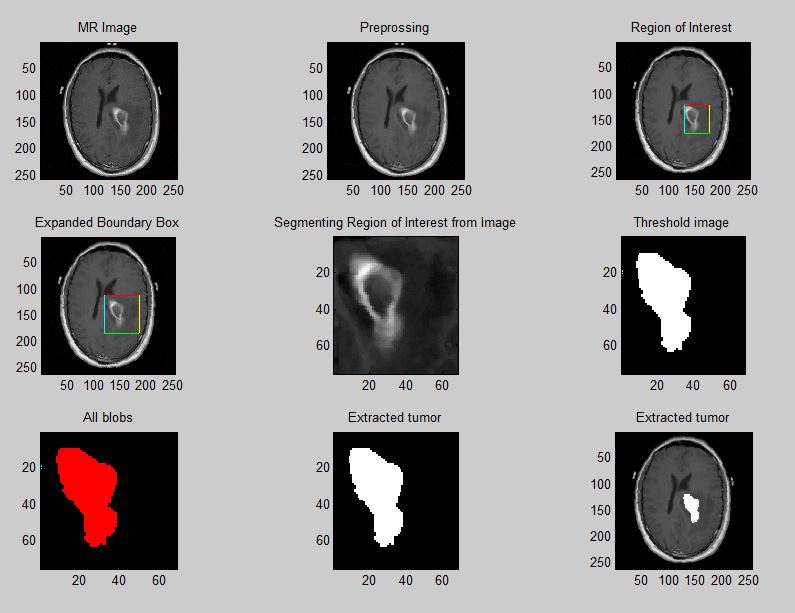


figure 1.1 Detection of tumor using image processing techniques.

* Security

Image processing has helped in developing efficient security/surveillance systems. Advancements in this field has impacted a lot of different consumer products as well as enterprises. Finger print unlock systems, Biometric security systems (face, iris recognition) are now being used in small devices like mobile phones to even smart buildings. With the use of these techniques unlocking devices has become simpler and easy compared to remembering and typing passwords or even carrying RFID security cards. These concepts have been extended to home security systems as well.

Work in the field of human body detection and recognition has led to smarter intrusion detection systems.

* Social Media  
  Various social media websites like Facebook, Instagram, Snapchat use some form of computer vision techniques to enhance the user experience. For example, Facebook’s auto tag feature recognizes faces in the pictures that users upload and suggests you an appropriate name tag for the person in the picture. Another application is Google image search. It searches for visually similar images over the world wide web which is a non-trivial task.

These are just some of the applications of computer vision (image processing). There are countless more such applications in the real world which are outside the scope of this book.

**Image processing libraries**

We will now introduce two image processing libraries scikit-image and pillow for python3 that will be used throughout this book to implement the algorithms that will be discussed. We will elaborate on how to install these libraries and show some basic image processing operations to prepare you for the next chapters.

**Pillow**

Pillow is an open source library that has been forked from the Python Imaging Library. Pillow is a very good starting point for beginners who want to start with implementing some basic algorithms before diving into the more complex ones. The book will use Pillow version 4.0

Note: https://python-pillow.org/

**Installation**

* **Windows**

Pillow can be installed on windows using pip.

Open Command Line tool on your windows machine and type in the following command and press enter.

pip install Pillow

Pip already comes installed with Python 2>=2.7.9 and Python 3>=3.4. In case you do not have pip installed please follow the official instructions given here: https://pip.pypa.io/en/stable/installing/#do-i-need-to-install-pip

* **OSX/macOS**

For OSX/macOS we will use Homebrew to install Pillow.

Go to <https://brew.sh/> for instructions on how to install Homebrew in case you do not have it installed

Install the dependencies using:

brew install libtiff libjpeg webp little-cms2

After that use pip to install Pillow

pip install Pillow

If you have both python2 and python3 installed, then to install Pillow for python3 use the following command

python3 –m pip install Pillow

* **Linux**

Use pip command to install

pip install Pillow

**Introduction to Pillow**

This section will walk you through basics of Pillow using relevant code snippets.

**Reading an image**

Knowing how to read an image is important. Almost all algorithms begin with an input image. In Pillow the Image module handles the read function (Image.open). This function returns an image object which contains information like image mode, image size and image format.

from PIL import Image

img = Image.open("image.png")

To display the image on your screen use the show() function

img.show()

**Writing/Saving Image**

To write or save an image to a file on your computer use the save() function associated to the image object. It takes in the absolute or relative file path to where you want to store the image.

img.save("temp.png")

**Cropping an image**

Cropping an image means to extract a particular region of the image which is smaller than the original image. This is sometimes useful when you want to run your algorithm only on a particular part of the image and not the entire image. This region in some books/references in called the Region of Interest (ROI). The image object has a crop() function that takes two coordinates – the upper left corner and the bottom right corner of the rectangle that you are interested in and returns the cropped image. This is illustrated in Figure 2.

from PIL import Image

dim = (100,100,400,400) #Dimensions of the ROI

crop\_img = img.crop(dim)

crop\_img.show()



Figure 2 (Left) Original Image. (Right) A cropped region of the original image

**Changing between color spaces**

Images can be converted from one mode to another using the convert function of the image module. To convert an image from RGB mode to grayscale mode use the “L” mode. There are various other modes available like “1” which is 1 bit pixel mode, “P” 8 bit pixel mode, “RGB” 3X8 bit pixel, “RGBA” 4X8 bit pixel.

NOTE: http://pillow.readthedocs.io/en/3.1.x/reference/Image.html#PIL.Image.Image.convert

from PIL import Image

grayscale = img.convert("L")

grayscale.show()



Figure 3 Output after converting from RGB mode to Grayscale

**Geometrical Transformation**

There are times when you need to perform multiple transformations to images like resize, rotate, flip and more. Pillow provides direct functions to perform these transformations saving you to write code from scratch.

**Resize**

To resize an image, use the resize() function which takes a tuple of new size as argument

from PIL import Image

resize\_img = img.resize((200,200))

resize\_img.show()

**Rotate**  
To rotate an image use the rotate() function which takes in the degrees to be rotated(Counter clockwise) as argument.

from PIL import Image

rotate\_img = img.rotate(90)

rotate\_img.show()



Figure 4 Output after rotating the image by 90 degrees.

**Image Enhancement**

Image enhancement are operations like changing contrast, brightness, color balance, sharpness of an image. Pillow provides an ImageEnhance module which has functions that can help you perform the earlier mentioned operations.

We will begin with importing the ImageEnhance module

from PIL import ImageEnhance

* Change brightness of an image

enhancer = ImageEnhance.Brightness(img)

enhancer.enhance(2).show()

The enhance function takes a float as an argument which describes the factor by which we want to change the brightness of the image. A factor value less that 1 will decrease the brightness and a factor value greater than 1 will increase the brightness of the image. Factor value equal to 1 will give the original image as output. The output of the enhance function is an image with the changed brightness.



Figure 5 This figure shows the increase in the brightness of the image. Left image is the original picture and the image on the right is the enhanced image.

* Change Contrast of the image:

enhancer = ImageEnhance.Contrast(img)

enhancer.enhance(2).show()

Again the enhance function takes a float argument. Factor equal to 1 will give you the original image and factor value less than 1 will decrease the contrast and greater than 1 will increase the contrast.



Figure 6 This figure shows the change in the contrast of the image. Left image is the original picture and the image on the right is the enhanced image.

**Accessing pixels of an image**

Sometimes there is a need to access individual pixels for performing tasks like thresholding (will be covered later in the book). Pillow provides a PixelAccess class with functions to manipulate image pixel values. getpixel(), putpixel() are some of the functions in the PixelAccess class.

* getpixel()

This function returns the color value of the pixel at the (x,y) coordinate. It takes a tuple as argument and returns a tuple of color values.

>>> img.getpixel((100,100))

output

(132, 56, 40)

Link to documentation: <link>

* putpixel()

This function changes the color value of the pixel at x, y coordinate to a new color value. Both the coordinates and the new color value is passed as an argument to the function. If the image has more than one band of colors then a tuple is passed as an argument to the function.

>>> img.putpixel((100,100),(0,0,0))

>>> img.getpixel((100,100))

output

(0, 0, 0)

**Link to documentation: <link>**

**Color Spaces and Channels**

Like in the world of mathematics we have different coordinate systems, for example a 2-D Cartesian Plane, 2-D Polar Coordinates. A point could be stored as (x,y) or (r, theta). Each coordinate system has a specific use-case which makes calculations easier. Similarly, in the world of image processing we have different color spaces. An image can store its color values in the form of Red, Blue, Green (RGB) or it could as Cyan, Magenta, Yellow, Key(black) (CMYK). Some examples of other color spaces are HSV, HSL, CMY and it goes on. Each value in the color space is called a color channel. For example, in the RGB color space we say that Red, Blue, Green each are channels of the image. Can you now calculate the number of channels CMYK will have? Yes, you got it right – it’s four. An image can be represented in many different modes (color spaces) like RGB, CMYK, Grayscale and Y’UV. The colors in the image that we see are derived by the mixture of the colors in each color channel of the color space. Let us look at some of the common color spaces in detail.

**Grayscale**

This is one of the simplest color spaces both in terms of understanding and storing on a computer. Each pixel value in a grayscale image is a single value between 0 and 255 with 0 representing black color and 255 representing white. Please keep in mind that the value 255 in not a fixed value but depends on the depth of the image (Image depth is covered in the section). Grayscale images are also sometimes called black and white images but it is not the entirely accurate. Black and white image means that the pixel values can only be either 0 or 255 and nothing in between them.



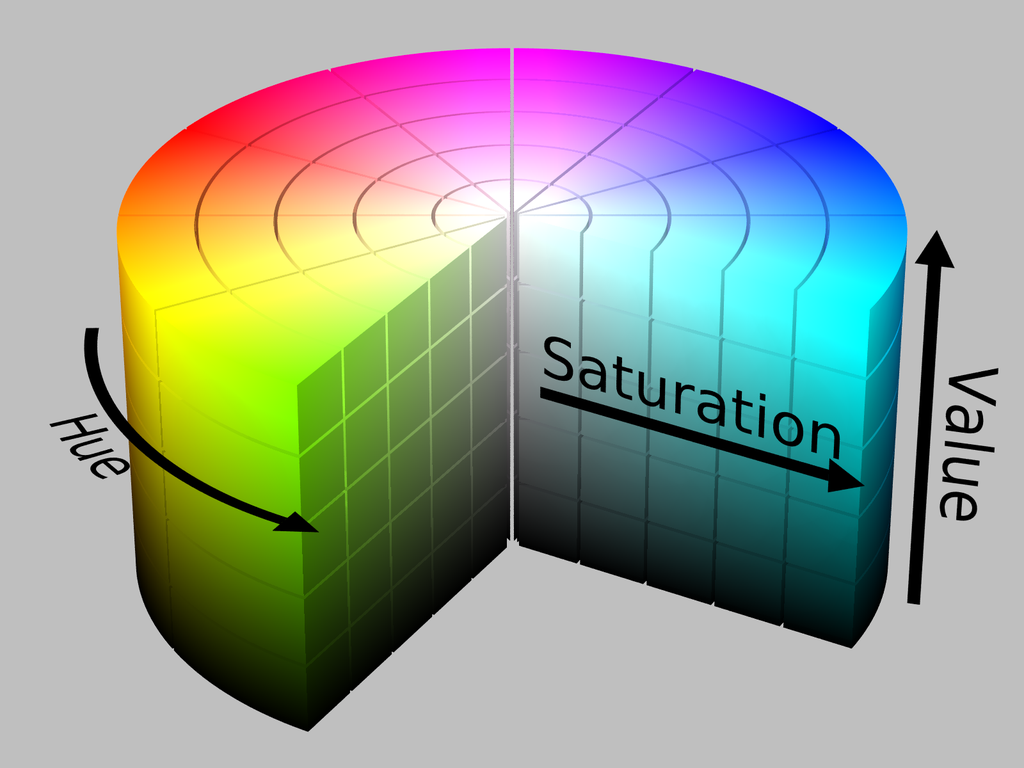
Figure 7 Example of a Grayscale image

**RGB (Red, Green, Blue)**

This is one of the most common color space that is used in the image processing world and otherwise. Most images that you view over the internet or in your books are in the RGB space. In a typical RGB image each pixel is a combination of three values each representing a color in Red, Green and Blue channels. White color in RGB space is written as (255, 255, 255) and Black is written as (0, 0, 0). Red, Green and Blue are represented by (255, 0, 0), (0, 255, 0) and (0, 0, 255) respectively. Any other color is just a combination of some values of Red, Green and Blue. Try to remember your painting class as a kid where you used to mix the primary colors to create a new color. That simple!

**HSV (Hue, Saturation, Value)**

This is a cylindrical coordinate system where we try to represent the RGB values in a cylindrical system. Figure 8 further illustrates this concept. The HSV color space was designed keeping mind the unintuitive nature of the RGB space. There is no clear intuition to how the color progress in the RGB space. HSV scale handles this perfectly in the sense that you can fix the Hue and then generate different shades of that hue by just varying values and saturation.



(By HSV\_color\_solid\_cylinder.png: SharkDderivative work: SharkD  Talk - HSV\_color\_solid\_cylinder.png, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=9801673)

At the beginning of the chapter we said, an image is stored in the form of a 2-D matrix. So how do we accommodate for the multiple channels in the image? Simple, we have multiple 2-D matrices for each channel. A little exercise – How many matrices will a grayscale image have?

If you try to print the pixel value of a grayscale image you will only get one value but if you try to print the pixel value of a RGB image then you will get three values, this shows that RGB has three channels red, green, blue and grayscale images have only one value.

>>> from PIL import Image

>>> img = Image.open("image.png")

>>> img.getpixel((100,100))

output

(132, 56, 40)

>>> img.convert("L").getpixel((100,100))

output

76

The following figure shows the different color channels in a RGB image.

Figure 9 Red, green, blue respectively.

**Image depth**

Image depth or the color depth is the number of bits used to represent a color of a pixel. The image depth determines the range of colors an image can have. For example if we have an image with depth 4 bits then the pixel value will range from 0 to 15. This means the highest number we can write using 4 bits. Whereas if we use 8 bits then the value will range from 0 to 255 providing a finer color spectrum. Another way of thinking about image depth is that the number of bits also determine the number of colors which can be used in an image. For example 1 bit implies 2 colors, 2 bits - 4colors and 8 bits - 256 colors and so on.

So far we have looked at only integer values for the colors. Some libraries also work with float images where the pixel value lies between 0 and 1. OpenCV is an example of one such library.

**Scikit-image**

In this section we are going to learn about another python library for image processing - scikit-image. Scikit-image provides more advanced operations as compared to Pillow and is suitable for building enterprise scale applications.

<Link> to scikit-image

**Installatios**

* **Linux**

sudo apt-get install python3-skimage

* **Windows**

pip3 install scikit-image

**Introduction to scikit-image (skimage)**

In this section we will walk through some basic operations that can be performed using scikit-image library.

**Reading an image**

As you know, reading an image is the most fundamental operations you would like to perform In scikit-image, image can be read using the imread() function in the io module of the library. It returns an ndarray. An ndarray in python is a N dimensional array.

>>> from skimage import io

>>> img = io.imread("image.png")

>>> io.imshow("image.png")

>>> io.show()

**Writing/Saving an image**

To save or write an image we can use the imsave() function. It takes the absolute or relative path of the file where you want to save the image and the image variable as input.

>>> from skimage import io

>>> img = io.imread("image.png")

>>> io.imsave(“new\_image.png”, img)

**Data module**

This module provides some standard test images which one can work on like a grayscale camera image, grayscale “text” image, coffee cup etc. these images can be used as great examples to demonstrate some of the algorithms in image processing.

For example, skimage.data.camera() returns an image array

>>> from skimage import data

>>> io.imshow(data.camera())

>>> io.show()

****

Figure 9 Image returned by the camera() function

skimage.data.text() returns an image array

>>> from skimage import data

>>> io.imshow(data.text())

>>> io.show()

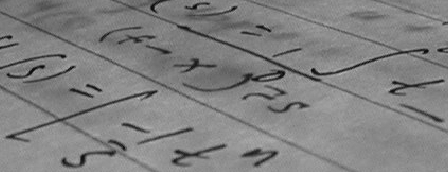


figure 10 image returned by text() function and it can used as an example for corner detection

**Color module**

This module of the library contains functions for converting image from one color space to another. Two of such functions are shown bellow:

**Convert RGB to Gray**

rgb2gray() function in the module can be used to convert a RGB image to a grayscale image. It takes the RGB image array as input and returns the grayscale image array.

>>> from skimage import io, color

>>> img = io.imread("image.png")

>>> gray = color.rgb2gray(img)

>>> io.imshow(gray)

>>> io.show()



figure 11

Convert RGB to HSV

rgb2hsv() function in the module can be used to convert a RGB image to an HSV image. It takes the RGB image array as input and returns the HSV image array.

>>> from skimage import color

>>> from skimage import data

>>> img = data.astronaut()

>>> img\_hsv = color.rgb2hsv(img)

**There are other functions which can be seem here: <link>**

**Draw module**

Draw module has various functions to draw different shapes like circle, line and polygon. Let us look at each of them one by one.

**Circle**

To draw a circle on an image, skimage provides a circle() function. It takes the centre coordinates and the radius as input and returns all the pixel coordinates which lie within the circle of the given coordinates and radius. After getting the pixels within the circle assign them the value 1 in the 2D matrix and all the other points make it 0. This would give you a circle.

>>> import numpy as np

>>> from skimage import io, draw

>>> img = np.zeros((100, 100), dtype=np.uint8)

>>> x , y = draw.circle(50, 50, 10)

>>> img[x, y] = 1

>>> io.imshow(img)

>>> io.show()

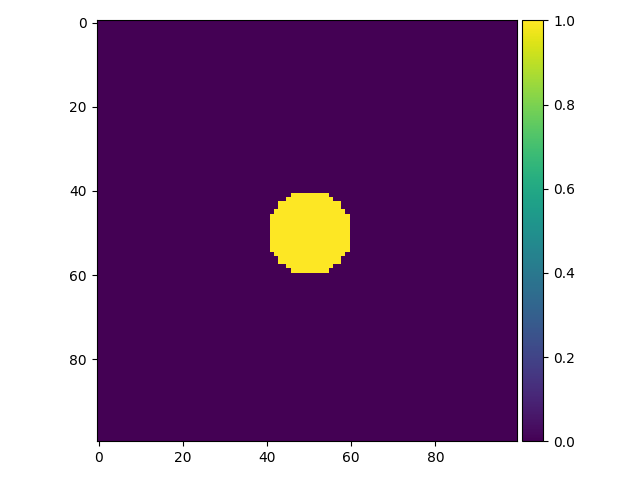


figure 12 circle of radius 10 and Centre (50, 50)

**Ellipse**

To draw an ellipse on an image, skimage provides an ellipse() function. This function of the draw module can be used to get the coordinates of the pixels within the ellipse of given parameters. Then these pixels can be distinguished from others by increasing the pixel value.

>>> import numpy as np

>>> from skimage import io, draw

>>> img = np.zeros((100, 100), dtype=np.uint8)

>>> x , y = draw.ellipse(50, 50, 10, 20)

>>> img[x, y] = 1

>>> io.imshow(img)

>>> io.show()

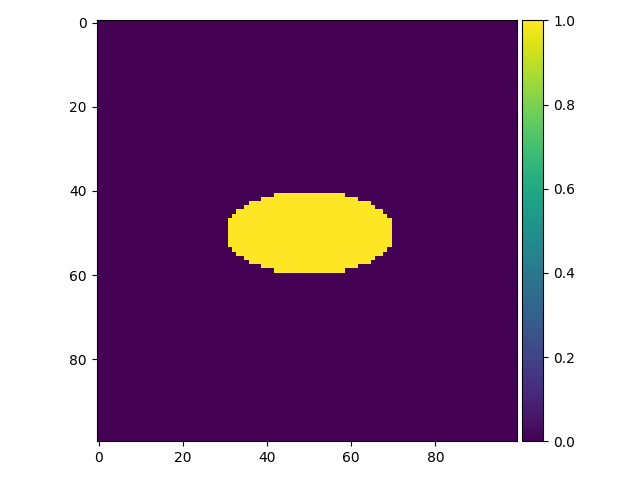


figure 13

**Polygon**

polygon() function takes the array of x and y coordinates of the vertices and returns the pixel coordinates which lie within the polygon.

>>> import numpy as np

>>> from skimage import io, draw

>>> img = np.zeros((100, 100), dtype=np.uint8)

>>> r = np.array([10, 25, 80, 50])

>>> c = np.array([10, 60, 40, 10])

>>> x, y = polygon(r, c)

>>> img[x, y] = 1

>>> io.imshow(img)

>>> io.show()

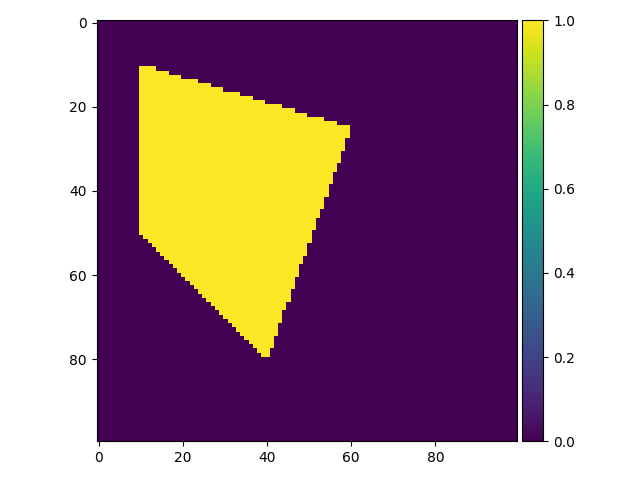


figure 14

## Conclusion

In this chapter we saw what Images are and how are they interpreted by a computer. Then we looked at the basic of image processing and its various applications in Medicine, Security/Surveillance and Social Media. Further two image processing libraries Pillow and Scikit-Image were introduced. We saw how we could perform basic operations like reading/writing an image. Converting the image between color spaces and finally we ended with how to draw some basic geometrical figures using scikit-image. This chapter forms the foundation of the chapters that follow. In the next chapter we will look at some more complex image processing algorithms like edge detection and also some commonly used filters.