

Introduction to Image Processing

Author Name: e-Yantra Team
ERTS Lab, IIT Bombay



Prerequisites: Fundamentals of Image Processing, Python Basics

Software required: Python 2.7.15, NumPy 1.14.5, OpenCV 2.4.13.6

Lecture Notes:

1. Introduction:

The field of digital image processing refers to processing digital images by means of a digital computer. It is used in a great number and wide variety of processes, these include processes from fault detection in assembly line operations, critical processes such as inspecting for microfissures (tiny cracks) in airplane fuselages and space-bound rockets, and tasks such as applying beautifying filters to photos in social media apps or catching traffic-light offenders via automated number-plate scanning.

2. Understanding of Concept:

What is an Image?

A digital image can be considered as a discrete representation of data possessing both **spatial (layout)** and **intensity (color)** information.

They can also be represented as a multidimensional signal. As such, in the two-dimensional case, they can be defined as, a function, $f(x, y)$, where x and y are **spatial (plane)** coordinates; and the amplitude of f at any pair of coordinates (x, y) is the **intensity (color)** of the image at that point.

They are predominantly 2D, but can be of higher dimensions as well (e.g. CT scans give 3D output). The fundamental discrete component in a 2D digital image is called a '**pixel**' and in a 3D image is called a '**voxel**'.

What is Image Processing?

Image processing is a form of signal processing for which the input is an image. In other words, it is a method of performing different operations on an image to get the features associated with it. It involves a wide variety of processes. One useful paradigm is to consider three types of computerized processes in this continuum: low-, mid-, and high-level processes.

- **Low-level processes** involve primitive operations such as image pre-processing to reduce noise, contrast enhancement, and image sharpening. A low-level process is characterized by the fact that both its inputs and outputs are images.

- **Mid-level processing** on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes or features extracted from those images (e.g., edges, contours, and the identity of individual objects).
- **High-level processing** includes tasks that involve “making sense” of a group of objects, for instance, it includes processes/algorithms for tasks such as object detection and face recognition.

How are images represented digitally?

Digital images are represented as **arrays** or **matrices**; which composes of rows and columns. These row and column dimensions, for e.g. **M rows** (ranging from **0 to $M-1$**) and **N columns** (ranging from **0 to $N-1$**) define the total number of pixels (**$M \times N$**) used to cover the digital depiction of the image. These rows and columns also give us the **spatial resolution** of the image, it is also called **digital resolution**.

An image has a spatial co-ordinate system unlike the conventional Cartesian co-ordinates. The **origin**, i.e., **(0, 0)** is located on **top-left corner** of an image, with **X-axis** spanning downwards from origin and **Y-axis** spanning towards right from origin, as shown in Figure 1.

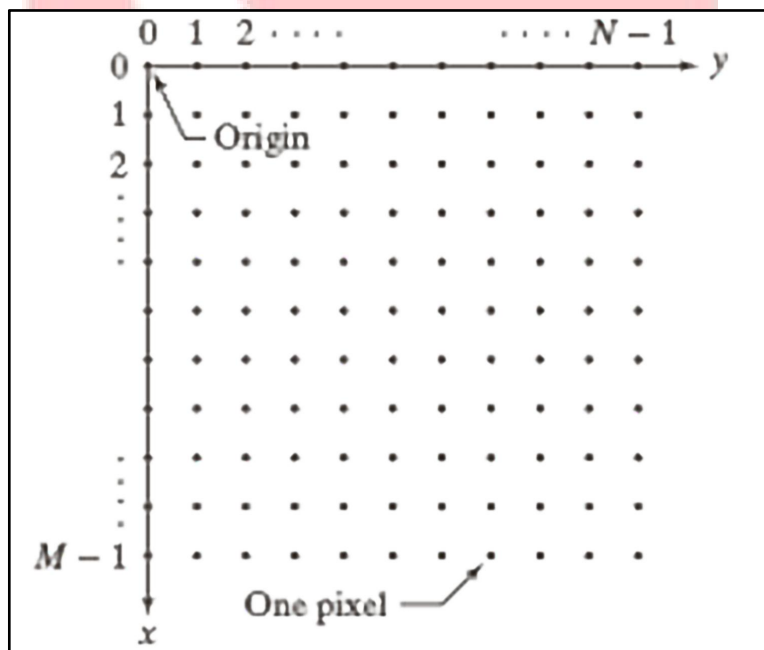


Figure 1: Digital representation of a 2D-image (courtesy: blogs.mathworks.com)

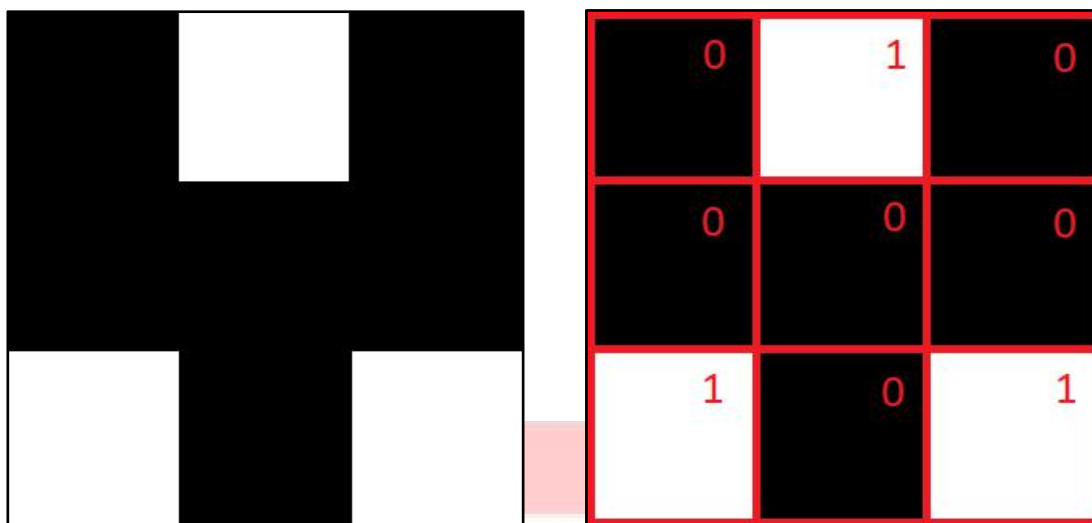


Figure 2(a): A 3 x 3 binary image

Figure 2(b): Values representing each pixel

In figure 2(a), we can see a simple digital image representing a character 'Y'. In order to represent it digitally we can set the matrix dimensions to be **3 rows** and **3 columns**. In case of a binary image, a pixel value of **0** is used to represent **black** and **1** is used to represent **white**, thus making it a **3 x 3 binary image**. The corresponding values of the pixels in Figure 2(a) can be seen in Figure 2(b). Here we can see that pixel value at origin, i.e. top-left pixel, $(0, 0) = 0$, value at pixel $(y, x) = (0, 1) = 1$, value at pixel $(y, x) = (1, 0) = 0$ and so on.

Images also have another type of resolution; namely the **bit resolution**. It is the number of binary bits required to represent a pixel in an image, for e.g. to represent a pixel in **binary** image **1 bit** is needed, and in **grayscale** image **8 bits** are needed.

The number of possible unique values (N) a pixel can have is given by a simple formula: $N = 2^k$, where k is the number of bits used to represent a pixel in image. The range of values a pixel can take is often referred to as the **dynamic range** of an image.

Hence, a pixel in **binary** image can have $N = 2^1 = 2$ values: **0** and **1**, and can only represent two different colors, typically: **black** and **white**.

On the other hand, a pixel in **grayscale** image can have $N = 2^8 = 256$ values, which correspond to different gray-level values. This can be observed in Figure 3(a) and Figure 3(b).

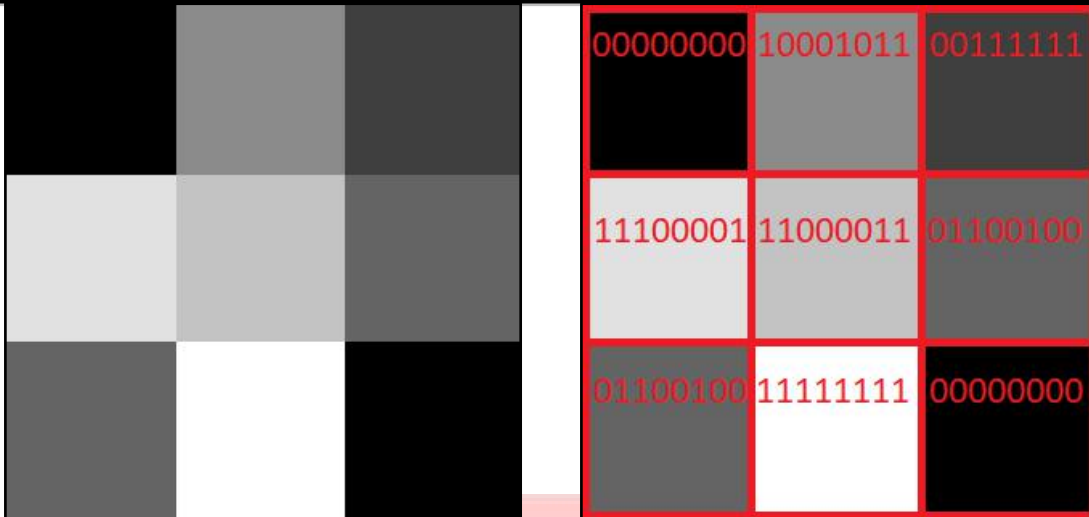


Figure 3(a): A 3 x 3 grayscale image

Figure 3(b): Values representing each pixel

Here, the pixel values at $(y, x) = (0, 1) = (10001011)_2 = (139)_{10}$ and at $(y, x) = (1, 2) = (01100100)_2 = (100)_{10}$ and so on.

Color images are represented with **3 channels/planes** of **8 bits** each, namely **Red (R)**, **Green (G)** and **Blue(B)**, so each pixel in such an image contains **24 bits** of information.

3. Installation and Setup:

Please refer to the installation and setup instructions provided in the document titled **Prerequisites for Image Processing.pdf**, to be downloaded from the Related Resources tab on the portal.

4. Commands / Functions:

Some elementary functions dealing with basic operations of an image such as reading, writing and displaying it would be introduced in this section.

Functions for Reading and Writing Images:

(a) `cv2.imread(filename[, flags]) → retval`

Use: To read or load an image from the specified image file and return the image as a multi-dimensional NumPy array based on its type.

Parameters:

- **filename:** path of image file along with filename and its extension as String; if the image is in the same folder as the Python program/script is, then only filename with extension is sufficient to specify as an argument
- **flags:** [optional] specifies the color type of a loaded image
 - ◆ **> 0** : returns a 3-channel color image
 - ◆ **= 0** : returns a grayscale image
 - ◆ **< 0** : returns the loaded image as is (with alpha channel)
 - ◆ **default : 1**, i.e. returns a 3-channel color image

For the supported image file formats and flags follow this [link](#).

- **retval:** returns an image as a multi-dimensional NumPy array for the image file specified

(b) `cv2.imwrite(filename, img[, params]) → retval`

Use: To save any multi-dimensional NumPy array or matrix as an image to a specified filename.

Parameters:

- **filename:** name of file along with its path; if image has to be saved in the folder or directory same as the Python script/program, then just the filename is sufficient to specify as an argument
- **img:** array or matrix variable from the Python program to be saved as an image
- **params:** [optional] these relate to the different file formats, quality of image to be saved, compression method to be used. By default (without any parameter), the lossless compression is used
- **retval:** returns **True** if the function is able to save the array or matrix as an image to a filename specified, else returns **False**

Note: The image format of the image to be saved is chosen based on the filename extension. Please refer this [link](#) for the list of extensions. Only 8-bit (or 16-bit in case of PNG, JPEG 2000 and TIFF) single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function. If the format, depth, or channel order is different, we need to use other functions like `cv2.cvtColor()` and `cv2.cvtColor()` to convert it before saving.

Functions for Displaying Images:

(c) `cv2.namedWindow(winname[, flags])` → None

Use: To create a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names. If a window with the same name already exists, the function does nothing.

Parameters:

- **winname:** name of the window in the window caption that may be used as a window identifier
- **flags:** [optional] flags of the window, the supported flags are:
 - ◆ **WINDOW_NORMAL** : if this is set, the user can resize the window (no constraint)
 - ◆ **WINDOW_AUTOSIZE** : if this is set, the window size is automatically adjusted to fit the displayed image and you cannot change the window size manually. If no flag is specified as an argument, this flag is set by default
 - ◆ **WINDOW_OPENGL** : if this is set, the window will be created with OpenGL support

To close the window and de-allocate any associated memory usage, functions `cv2.destroyWindow()` or `cv2.destroyAllWindows()` can be called.

(d) `cv2.imshow(winname, img)` → None

Use: To display an image in the specified window.

Parameters:

- **winname:** name of the window where the image matrix or array is to be displayed
- **img:** array or matrix variable from the Python program to be displayed as an image

If the window was created with the **WINDOW_AUTOSIZE** flag, the image is shown with its original size, however it is still limited by the screen resolution. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth, please refer this [link](#) for more details.

If the window was not created before this function, it is assumed creating a window with **WINDOW_AUTOSIZE** flag set.

If you need to show an image that is bigger than the screen resolution, you will need to call **cv2.namedWindow()** function with **WINDOW_NORMAL** flag set before the **cv2.imshow()**.

Note: If the program ends at this function, the image to be displayed will just flash for once on the screen. In order to display the image window until any key is pressed or for a specific amount of time, this **cv2.imshow()** function should be followed by the **cv2.waitKey()** function described below.

(e) cv2.waitKey([delay]) → retval

Use: To wait for a pressed key.

Parameters:

- **delay:** delay in milliseconds. **0** is the special value that means ‘forever’
- **retval:** the code of the pressed key or **-1** or if no key was pressed before the specified time had elapsed

The function waits for a key event infinitely (when **delay ≤ 0**) or for **delay** milliseconds, when it is positive.

(f) cv2.destroyAllWindows(winname) → None

Use: To close or destroy a window and de-allocate any associated memory usage of the given name.

Parameters:

- **winname:** name of the window to be closed or destroyed

(g) cv2.destroyAllWindows() → None

Use: To close or destroy all of the opened windows and de-allocate the associated memory usage for all the windows.

5. Program

Steps to write a program:

1. Open “IDLE (Python GUI)” application
2. Select File → New
3. Type your program.

Problem Statement:

Write a program to execute the following steps:

- (i) Load or read a color image file (**‘color_image.jpg’** is provided under Related Resources tab on portal) in a variable. Read the same image as a grayscale image in another variable.
- (ii) Examine and print the shape of both the resultant NumPy arrays.
- (iii) Create a window named ‘color image’ and display color image in this window until any key is pressed.
- (iv) Display grayscale image for next 5 seconds, no need to create separate window.
- (v) Write or save the grayscale image to a file in the current directory.

Note: Instead of using a saved image, if you want to read an image from an attached camera such as a webcam, refer to the document titled, **Capturing Image from Camera.pdf** to know more about the functions, their usage and an example. This document is to be downloaded from the Related Resources tab on the portal.

Program Snippet:

```
# import OpenCV and NumPy
import cv2
import numpy as np

# load or read a color image file into a variable
# here, it is assumed that the image file is in the same
# directory as this Python script/program, hence only name of
# image file is provided as an argument to the function
color_img = cv2.imread('color_image.jpg')

# color_img = cv2.imread('color_image.jpg', 1) -- this is same
# as the above statement without any flags specified

# load or read the same color image file as grayscale
# into another variable
gray_img = cv2.imread('color_image.jpg', 0)
```

```
# 'shape' attribute of NumPy array returns its dimensions
# in case of images, format is: (height, width, channels);
# if channels = 3, then the image is colored image having
# Red, Green and Blue channels; while grayscale image has
# only one channel, hence 'shape' will return height and width
print "Color Image shape = ", color_img.shape
print "Grayscale Image shape = ", gray_img.shape

# create window for color image
cv2.namedWindow('color image')

# display color image in 'color image' window
cv2.imshow('color image', color_img)

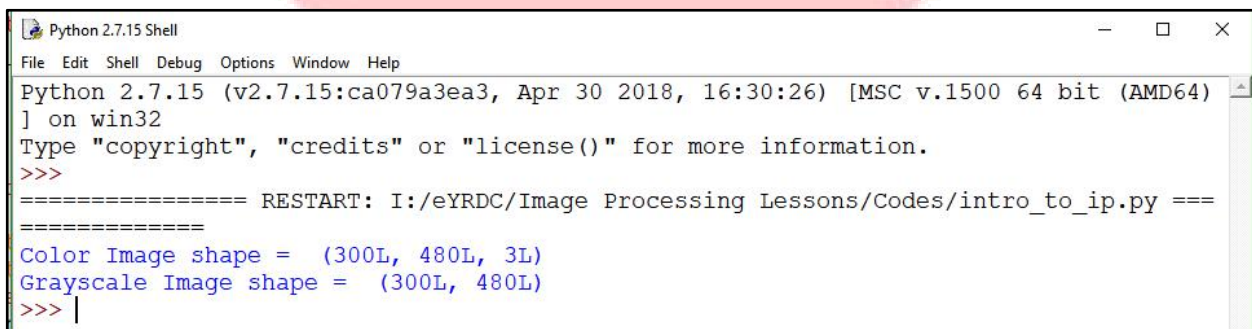
# display color image until any key is pressed, once a key is
# pressed, destroy or close only the color image window
cv2.waitKey(0)
cv2.destroyWindow('color image')

# display grayscale image
cv2.imshow('grayscale image', gray_img)

# wait for only 5 seconds, then destroy all windows
cv2.waitKey(5000)
cv2.destroyAllWindows()

# write or save the grayscale image in the current directory
cv2.imwrite('grayscale image.jpg', gray_img)
```

Output:



```
Python 2.7.15 Shell
File Edit Shell Debug Options Window Help
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: I:/eYRDC/Image Processing Lessons/Codes/intro_to_ip.py =====
Color Image shape = (300L, 480L, 3L)
Grayscale Image shape = (300L, 480L)
>>> |
```

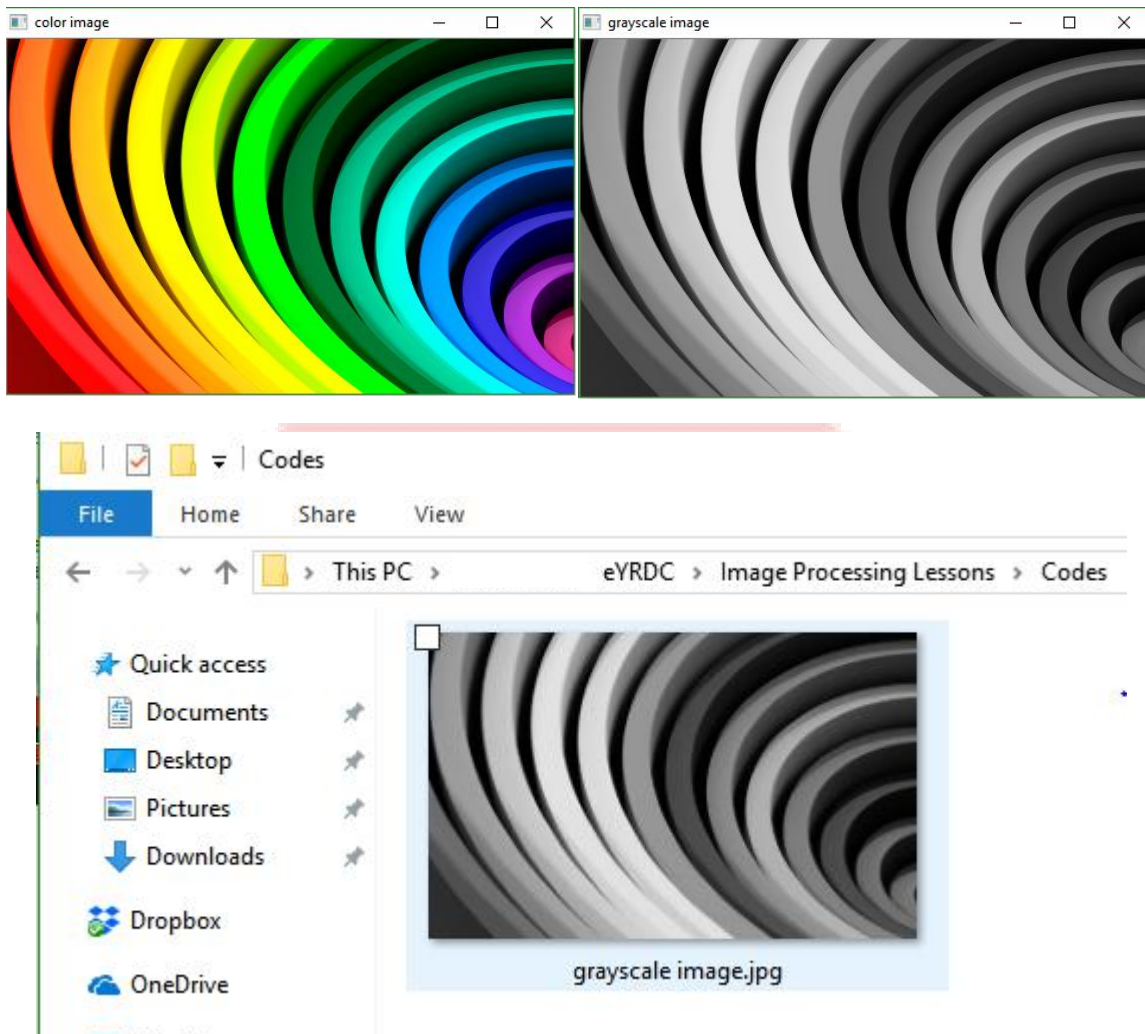


Figure 4: Output of the sample program

References

- [1]. C. Solomon, T. Breckon, "Fundamentals of Digital Image Processing", 1st ed., Wiley-Blackwell: 2011, pp.1-4
- [2]. R. Gonzalez, R. Woods, "Digital Image Processing", 2nd ed., Prentice Hall:2002, pp.15-16
- [3]. <https://docs.opencv.org/2.4/index.html>