**National University of Computer and Emerging Sciences**



Lab Manual 09

AL2002-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Hajra Waheed |
| Lab Instructor (s) | Hamza Ayub  Muhammad Ahmed |
| Sections | B1 - B2 |
| Semester | Spring 2023 |

**Department of Computer Science**

**FAST-NU, Lahore, Pakistan**

# Introduction to Image Processing

We examine exactly what an image is, and its related properties. You should have an understanding about the following concepts:

* + Images
  + Pixels
  + Image resolution

### Images

Visual representation of a real-life object (a person or any other object) in a two-dimensional form is called an *image*. An image is nothing but a collection of pixels in different color spaces. Here is an example of a normal image.



### Pixels

You might think of a complete image as a set that consists of small samples. These samples are called *pixels*. They are the smallest elements in any digital image. Have you ever zoomed in on an image to such an extent that you see small squares? Those are pixels. So, pixels are subsamples of an image that, when get combined, give us the complete image. See how pixels, with various colors, may look.



*Pixels of various colors (Source:* [*www.freeimages.co.uk*](http://www.freeimages.co.uk/)*)*

### Image Resolution

Image resolution is the number of pixels present in an image. The greater the number of pixels, the better quality. Image resolutions are described, for example, as 320 × 240, 640 × 480, 800 × 600, 1024 × 768, and so on.

This means, for example, that there are 1024 pixel columns and 768 pixel rows. The total number of pixels is obtained by multiplying both numbers, which gives us 786,432 pixels. Figure below shows comparative depictions of different image resolutions.

640x480

1280x720

1920x1080

*Comparative image resolution (Source:* [*www.*](http://www.freeimages.co.uk/)[*freeimages.co.uk*](http://www.freeimages.co.uk/)*)*

### Scikit Image

Scikit Image is a module that is used to do basic image processing. Before we start, let’s look at the definition of a module. A module is a collection of Python files, classes, or functions. We can save complex and lengthy code inside different files. To do this, we need to import the files and use them in our environment. First we need to import Scikit Image into our environment, like so:

import skimage

This single line of code imports an entire collection of classes and functions needed to do basic image analysis.

In this section we look at following operations using Scikit Image and Python:

* Uploading and Viewing an Image
* Getting Image Resolution
* Looking at Pixel Values
* Converting Color Space
* Saving an Image
* Creating Basic Drawings
* Doing Gamma Correction
* Rotating, Shifting, and Scaling Images
* Determining Structural Similarity

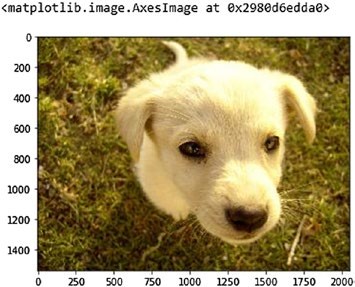
#### Uploading and Viewing an Image

Let’s see how we can import an image into the Python environment and view it there. We start by importing a module named skimage, which contains different image processing algorithms. To upload and view the image, we use a class from the skimage module called io. Inside this class, we use the imread function to upload and read an image; the function imshow is used to view the image. Let’s have a look at the code.

from skimage import io

img = io.imread('puppy.jpg') io.imshow(img)

Output:



#### Getting Image Resolution

To get the resolution of the image, we use a built-in function called shape. When an image is read, all the pixel values are stored in an array format; this array is called a *numpy array*. After we read the image and convert it to array, we use the shape function to look at the resolution.

In the following code, you can see that we have an image with a resolution of 1536 × 2048, and it has three channels (because it is in the RGB color format).

#Getting Image Resolution from skimage import io

img = io.imread('puppy.jpg') img.shape

Output:

(1536, 2048, 3)

#### Looking at Pixel Values

Now that we know the resolution of the image, we may want to look at each pixel value. To do this, we save the numpy array in one line—in other words, we use one row to store all the pixel values. When you look at he code that follows, you can see we are importing another module named pandas. Pandas is used to read, write, and process various file formats.

Here, we save the pixel values in the Excel format:

#Getting Pixel Values from skimage import io import pandas as pd

img = io.imread('puppy.jpg')

df = pd.DataFrame(img.flatten()) filepath = 'pixel\_values1.xlsx' df.to\_excel(filepath, index=False)

When we look at the importing line—import pandas as pd—it means we are renaming the imported module to pd. The flatten function is used to convert the three dimensions of an RGB image to a single dimension.

We then save that image in an excel file named pixel\_values.xlsx. To do this, we use the Pandas function called to\_excel. The DataFrame function converts a one-dimensional array into an Excel-like format, with rows and columns. You can print the df variable to look at the data frame structure.

#### Converting Color Space

Suppose our image is in the RGB color space. We may want to convert it to different color formats. In this section we look at different conversions, then convert the image back to its original RGB format.

Before looking at the code, we must examine the functions we will use.

For converting an image into different color formats, we need to use the

class color, which is present in skimage module. Inside this class, we can use the following functions:

* rgb2hsv
* hsv2rgb
* rgb2xyz
* xyz2rgb
* rgb2grey
* grey2rgb
* rgb2yuv
* yuv2rgb
* rgb2lab
* lab2rgb
* rgb2yiq
* yiq2rgb
* rgb2ypbpr
* ypbpr2rgb

Also, we have to use one more module, called pylab. We import all the classes present inside pylab by using \*. We use pylab to see different figures in different blocks. Then we use the function figure to display more than one image at a go. Let’s now look at all the code and its output.

##### RGB to HSV and Vice Versa

#Import libraries

from skimage import io from skimage import color from skimage import data from pylab import \*

#Read image

img = io.imread('puppy.jpg')

#Convert to HSV

img\_hsv = color.rgb2hsv(img)

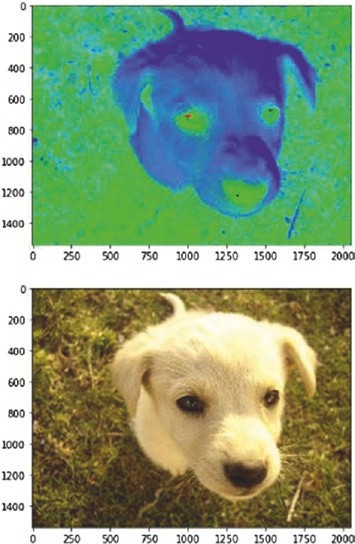
#Convert back to RGB

img\_rgb = color.hsv2rgb(img\_hsv)

#Show both figures figure(0)

io.imshow(img\_hsv) figure(1) io.imshow(img\_rgb)

Output:



##### RGB to XYZ and Vice Versa

#Import libraries

from skimage import io from skimage import color from skimage import data #Read image

img = io.imread('puppy.jpg')

#Convert to XYZ

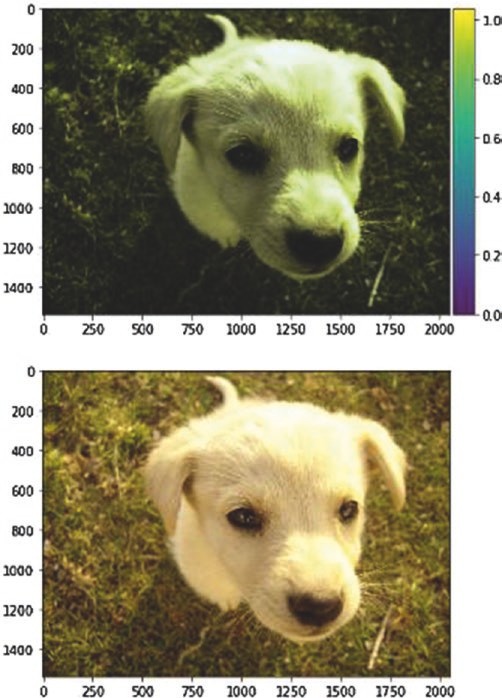
img\_xyz = color.rgb2xyz(img)

#Convert back to RGB

img\_rgb = color.xyz2rgb(img\_xyz)

#Show both figures figure(0) io.imshow(img\_xyz) figure(1) io.imshow(img\_rgb)

Output:



##### RGB to LAB and Vice Versa

#Import libraries

from skimage import io from skimage import color #Read image

img = io.imread('puppy.jpg')

#Convert to LAB

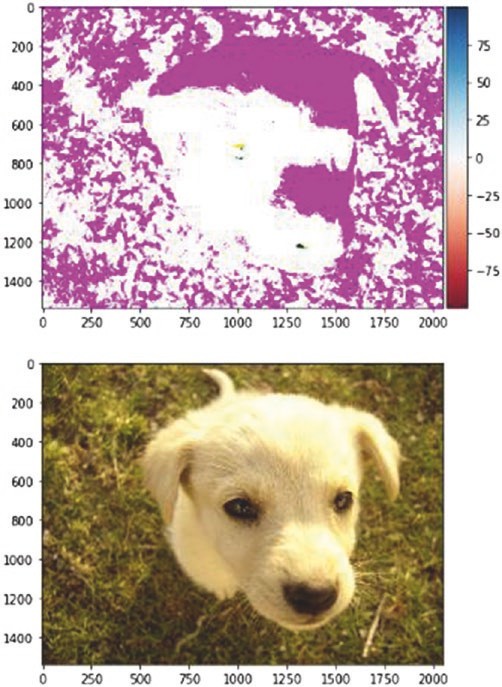
img\_lab = color.rgb2lab(img)

#Convert back to RGB

img\_rgb = color.lab2rgb(img\_lab)

#Show both figures figure(0) io.imshow(img\_lab) figure(1) io.imshow(img\_rgb)

Output:



##### RGB to YUV and Vice Versa

#Import libraries

from skimage import io from skimage import color #Read image

img = io.imread('puppy.jpg')

#Convert to YUV

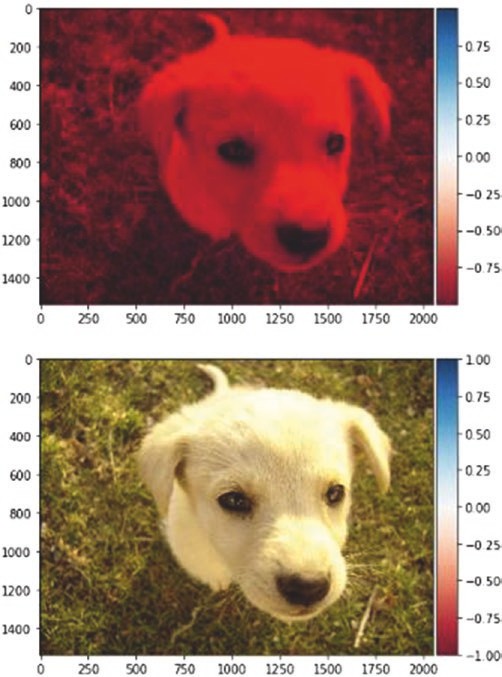
img\_yuv = color.rgb2yuv(img)

#Convert back to RGB

img\_rgb = color.yuv2rgb(img\_yuv)

#Show both figures figure(0) io.imshow(img\_yuv) figure(1) io.imshow(img\_rgb)

Output:



##### RGB to YIQ and Vice Versa

#Import libraries

from skimage import io from skimage import color #Read image

img = io.imread('puppy.jpg')

#Convert to YIQ

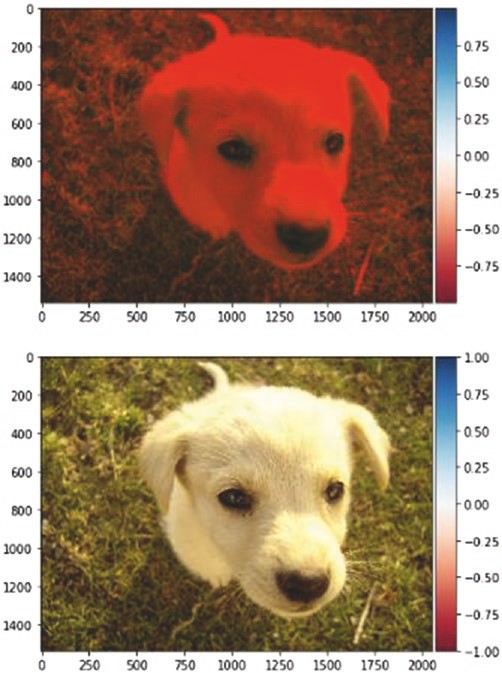
img\_yiq = color.rgb2yiq(img)

#Convert back to RGB

img\_rgb = color.yiq2rgb(img\_yiq)

#Show both figures figure(0) io.imshow(img\_yiq) figure(1) io.imshow(img\_rgb)

Output:



##### RGB to YPbPr and Vice Versa

#Import libraries

from skimage import io from skimage import color #Read image

img = io.imread('puppy.jpg')

#Convert to YPbPr

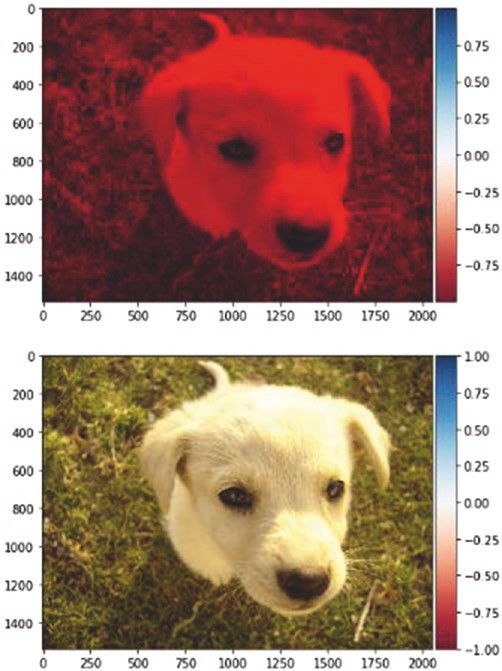
img\_ypbpr= color.rgb2ypbpr(img)

#Convert back to RGB

img\_rgb= color.ypbpr2rgb(img\_ypbpr)

#Show both figures figure(0) io.imshow(img\_ypbpr) figure(1) io.imshow(img\_rgb)

Output:



#### Saving an Image

After every image analysis, we may want to save the image. To do this, we use the skimage io function called imsave. In the following code, the

first argument includes the name of the file to which you want to save the image; the second is the variable that contains the image.

#Import libraries

from skimage import io from skimage import color from pylab import \*

#Read image

img = io.imread('puppy.jpg')

#Convert to YPbPr

img\_ypbpr= color.rgb2ypbpr(img)

#Convert back to RGB

img\_rgb= color.ypbpr2rgb(img\_ypbpr) io.imsave("puppy\_ypbpr.jpg", img\_ypbpr)

#### Creating Basic Drawings

Within an image, we might like to draw certain figures. These figures can be simple, such as a line, or complex, such as an ellipsoid. Let’s look at some basic drawings using the skimage drawing class called draw.

##### Lines

The line function is used to draw a simple line on an image. In the following code, the first two parameters indicate the first point; the last two parameters indicate the second point. A line is then drawn using these points. We can then change the pixel values of the line so we are able to see the line on the image.

from skimage import io from skimage import draw

img = io.imread('puppy.jpg') x,y = draw.line(0,0,1000,1000) img[x, y] = 0

io.imshow(img) Output:



##### Rectangles

To draw rectangles, we use the function polygon. We can draw not only rectangle, but any kind of polygon we want. All we have to do is give x and y coordinates, then define the width and the height.

In the following code, I use the function rectangle. It returns a shape with pixel values that we change, as in the previous example of a line.

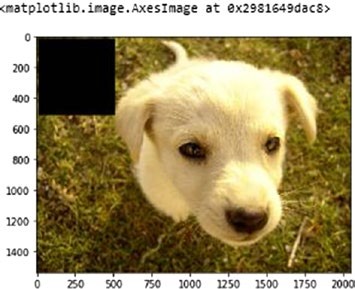
from skimage import io from skimage import draw

img = io.imread('puppy.jpg') def rectangle(x, y, w, h):

rr, cc = [x, x + w, x + w, x], [y, y, y + h, y + h]

return (draw.polygon(rr, cc))

rr, cc = rectangle(10, 10, 500,500) img[rr, cc] = 1

io.imshow(img) Output:

##### Circles

The circle function is used to draw a circle. In the following code, the first two arguments indicate the position of the circle inside the image; the last argument indicates the radius.

#Import libraries

from skimage import io from skimage import draw

#Load image

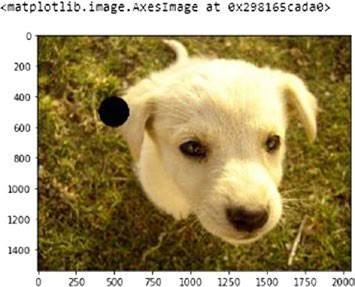
img = io.imread('puppy.jpg')

#Define circle coordinates and radius x, y = draw.circle(500,500, 100)

#Draw circle img[x, y] = 1

#Show image io.imshow(img)

Output:



##### Bezier Curve

To draw a Bezier curve, we using the function bezier\_curve. We need to indicate the position of three or more control points that then shape the curve. The first six arguments in the following code define three points; the last argument defines the tension present in the line. Play with different values change the curve.

#Import libraries

from skimage import io from skimage import draw

#Load image

img = io.imread('puppy.jpg')

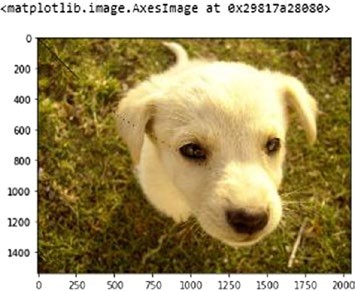
#Define Bezier curve coordinates

x, y = draw.bezier\_curve(0,0, 500, 500, 900,1200,100) #Draw Bezier curve

img[x, y] = 1

#Show image io.imshow(img)

Output:



#### Doing Gamma Correction

To perform gamma correction of an image, based on the display instrument, we use exposure class in skimage. The exposure class contains a function called adjust\_gamma, which we use to give an image as an input and the final gamma value that we want. In this way, we get a gamma-corrected image.

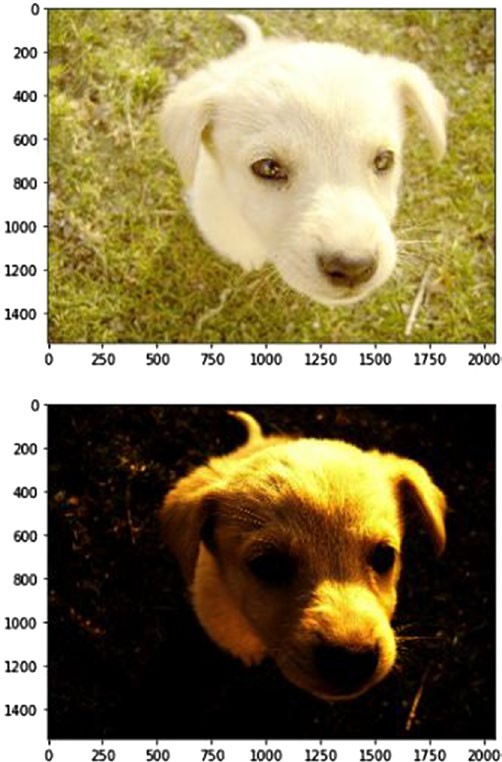
from skimage import exposure from skimage import io

from pylab import \*

img = io.imread('puppy.jpg')

gamma\_corrected1 = exposure.adjust\_gamma(img, 0.5) gamma\_corrected2 = exposure.adjust\_gamma(img, 5) figure(0)

io.imshow(gamma\_corrected1) figure(1)

io.imshow(gamma\_corrected2) Output:

### Rotating, Shifting, and Scaling Images

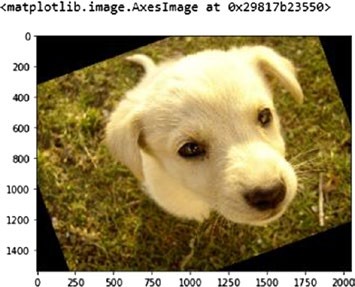
Sometimes we may want to rotate an image or change its size. To do this, we use the transform class in the skimage module. transform has two functions: rotate and resize. rotate takes the degree of rotation as its parameter; resize takes the new size as its parameter.

from skimage import io

from skimage.transform import rotate img = io.imread('puppy.jpg')

img\_rot = rotate(img, 20) io.imshow(img\_rot)

Output:

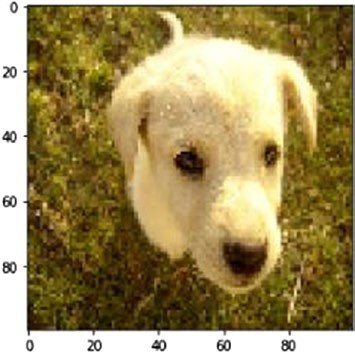


from skimage import io

from skimage.transform import resize img = io.imread('puppy.jpg')

img\_res = resize(img, (100,100)) io.imshow(img\_res) io.imsave("ss.jpg", img\_res)

Output:



### Determining Structural Similarity

As we explained earlier, structural similarity is used to find the index that indicate how much two images are similar. A value closer to one means the images are very similar; a value closer to zero means they are less similar.

In the following code, for the first comparison of similar images, we get a SSIM output of 1.0. In the second bit of code, in which we compare the image with its YPbPr counterpart, we get a SSIM out of 0.43, which indicates less similarity.

from skimage import io

from skimage.measure import compare\_ssim as ssim img\_original = io.imread('puppy.jpg') img\_modified = io.imread('puppy\_ypbpr.jpg')

ssim\_original = ssim(img\_original, img\_original, data\_range=img\_ original.max() - img\_original.min(), multichannel=True) ssim\_different = ssim(img\_original, img\_modified, data\_range=img\_ modified.max() - img\_modified.min(), multichannel=True) print(ssim\_original,ssim\_different)

Output:

1.0 0.4348875243670361

SSIM takes three arguments. The first refers to the image; the second indicates the range of the pixels (the highest pixel color value less the lowest pixel color value). The third argument is multichannel. A True value means the image contains more than one channel, such as RGB. False means there is only one channel, such as grayscale.

**Exercise Question:**

Write a Python function that takes an input image and applies the following operations using scikit-image library:

* Load Image
* Convert the image to grayscale.
* Apply all the image preprocessing techniques as mentioned in the notebook on given image