- **1. Introduction to Digital Images and Image Processing**

Welcome to this tutorial on image processing with Python! In this tutorial, we will explore the world of image processing and how it can be accomplished using the Python programming language. While we will cover multiple libraries and frameworks, one that we will frequently utilize is the OpenCV library.

Image processing is a powerful technique that enables us to manipulate digital images, enhancing them or extracting valuable information. By applying various algorithms and methods, we can achieve desired results and uncover insights from images.

Python is an ideal language for image processing due to its simplicity and the availability of numerous libraries and frameworks. Among these, OpenCV (Open Source Computer Vision Library) is one of the most popular choices. OpenCV offers a rich set of functions and tools for image and video analysis, manipulation, and processing.

Throughout this tutorial, we will delve into different aspects of image processing using Python and OpenCV. We will explore techniques such as image filtering, edge detection, image transformations, object detection, and more. Moreover, we will discuss various practical applications where image processing proves invaluable.

By the end of this tutorial, you will have a solid understanding of how to perform image processing tasks using Python and OpenCV. This knowledge will empower you to apply these techniques effectively in your own projects.

Let's embark on this exciting journey into the world of image processing!

1.1 What is Image Processing

Understanding the concept of image processing is essential before delving into its implementation. Image processing, also known as digital image processing, plays a crucial role in the field of computer vision. While these terms may sound similar, it's important to comprehend their relationship.

In image processing, algorithms are applied to digital images, transforming them to enhance their quality or extract specific information. The output of image processing remains an image, which can undergo various modifications. On the other hand, computer vision algorithms analyze images to derive features or obtain information about the content of the image.

In summary, image processing focuses on manipulating and improving images, while computer vision aims to extract meaningful insights or features from those images. Both domains have their unique applications and are interconnected in the broader field of visual analysis.

Now that we have a clear understanding of image processing and its relationship with computer vision, we can proceed to explore its applications and benefits.

1.2 Why do we need it

Raw data that we collect or generate often requires preprocessing before it can be effectively used in applications. This preprocessing involves analyzing the data and performing necessary transformations to make it suitable for further use.

For example, let's consider the scenario of building a cat classifier. To train the classifier, we would need a dataset consisting of hundreds of cat images. However, the images collected may not have the same size or dimensions. To address this issue, we need to preprocess the images by resizing them to a standard size before feeding them into the model for training.

This example highlights the importance of image processing in computer vision applications. Image processing allows us to address various challenges associated with raw image data, such as

standardizing sizes, adjusting brightness or contrast, removing noise, and much more. By applying appropriate image processing techniques, we can improve the quality of the data and extract meaningful features, making it easier for computer vision models to analyze and interpret the images accurately.

Image processing plays a vital role in preparing data for computer vision tasks and is a fundamental step in developing robust and accurate vision-based applications.

1.3 Prerequisites

Before we proceed, let's discuss the prerequisites for following this tutorial effectively. It is recommended to have some basic programming knowledge in any language as a foundation. Additionally, familiarity with the concept of machine learning and its fundamentals is beneficial, as we will be utilizing machine learning algorithms for image processing in this tutorial. However, prior exposure to OpenCV is not required, although it can be helpful.

One crucial prerequisite for understanding this tutorial is knowing how an image is represented in memory. An image is essentially a collection of pixels, which are stored as a matrix of pixel values. In the case of grayscale images, the pixel values range from 0 to 255, representing the intensity of each pixel. For example, a grayscale image with dimensions 20x20 would be represented by a 20x20 matrix, comprising a total of 400 pixel values.

When dealing with colored images, it's important to understand that they have three channels: Red, Green, and Blue (RGB). Consequently, a single image will have three corresponding matrices. These matrices capture the pixel values for each color channel, forming the complete representation of the colored image.

Having a grasp of these concepts will enable you to follow along smoothly as we explore various image processing techniques in Python.

Now that we have covered the prerequisites, let's dive into the exciting world of image processing with Python!

1.4 Installation

Note: Since we are going to use OpenCV via Python, it is an implicit requirement that you already have Python (version 3) already installed on your workstation.

Windows

$ pip install opencv-python

MacOS

$ brew install opencv3 --with-contrib --with-python3

Linux

$ sudo apt-get install libopencv-dev python-opencv

To check if your installation was successful or not, run the following command in either a Python shell or your command prompt:

import cv2

1.5 Implementation

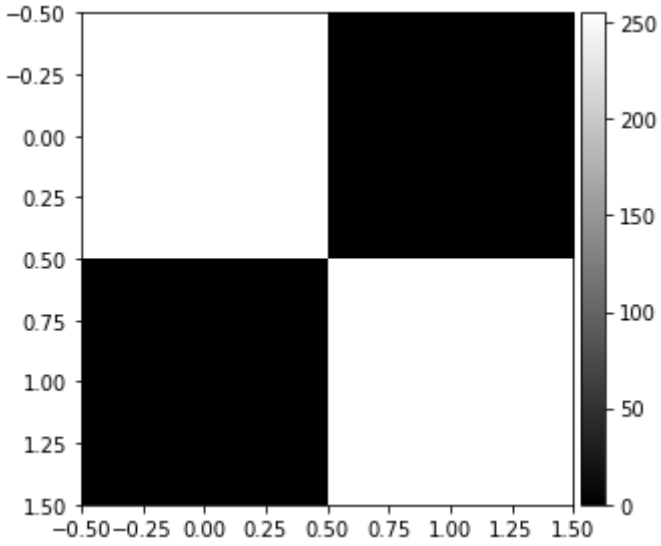
Image Processing using Python is a fun way to understand how pictures can be represented via math and code. I hope that this short article can give you an idea of how your machines understand image data.

Let’s get started.

```
# First import the required Python Libraries
import numpy as np
import matplotlib.pyplot as pltfrom skimage import
img_as_uint
from skimage.io import imshow, imread
from skimage.color import rgb2hsv
from skimage.color import rgb2gray
```

Great! Now that we have imported the required libraries, we can begin with the basics. An image can be thought of as a matrix where every pixel’s color is represented by a number on a scale.

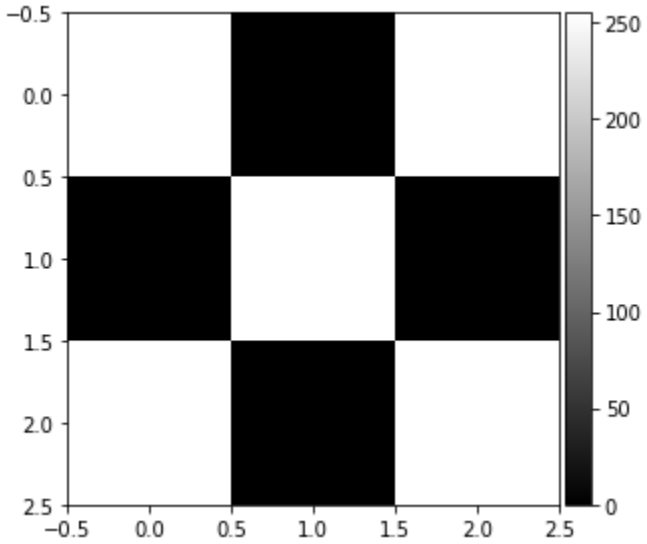
```
array_1 = np.array([[255, 0],
                    [0, 255]])imshow(array_1, cmap =
'gray');
```



Output 1

The above output is a visual representation of the matrix we just created. Note that we are not limited to a simple 2x2 matrix. Below is an example of a 3x3 matrix.

```
array_2 = np.array([[255, 0, 255],
                    [0, 255, 0],
                    [255, 0, 255]])
imshow(array_2, cmap = 'gray');
```



Output 2

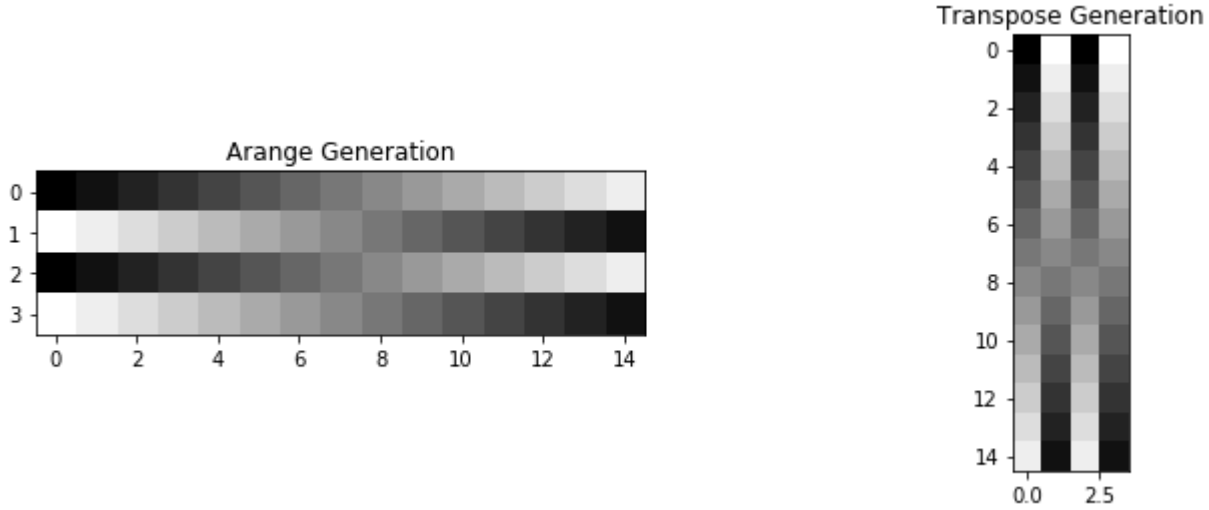
Although our examples were picking colors on the extreme end of the spectrum, we can also access any of the colors in between.

Remember that you do not have to manually encode each number!

The below image was constructed with the use of the *arange* function of NumPy and created another image by getting the *transpose* of the first one.

```
array_spectrum = np.array([np.arange(0,255,17),
                   np.arange(255,0,-17),
                   np.arange(0,255,17),
                   np.arange(255,0,-17)])fig, ax =
plt.subplots(1, 2,
figsize=(12,4))ax[0].imshow(array_spectrum, cmap
= 'gray')
```

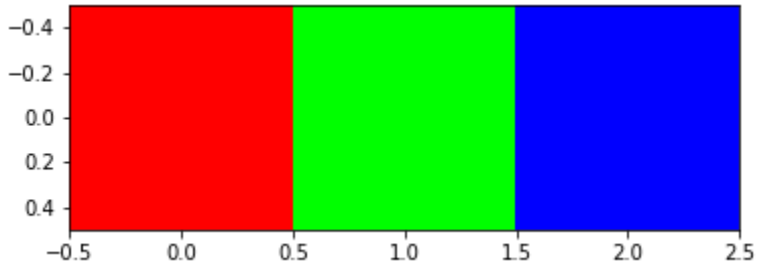
```
ax[0].set_title('Arange
Generation')ax[1].imshow(array_spectrum.T, cmap
= 'gray')
ax[1].set_title('Transpose Generation');
```



Output 3

For simplicity we have been using grayscale, but please remember that the machine actually understands colors as different mixtures of Red, Green, and Blue. As such, we can represent images as 3 dimensional matrixes. Each pixel being represented as a Python list that specifies their color mix.

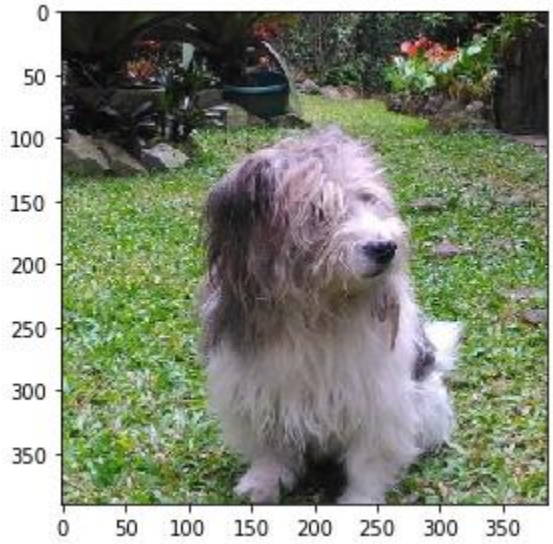
```
array_colors = np.array([[[255, 0, 0],
                          [0, 255, 0],
                          [0, 0, 255]]])imshow(array_colors);
```



Output 4

Let’s try manipulating an actual image. Below is an image of a loveable doggo.

```
doggo = imread('doggo.png')
imshow(doggo);
```



Doggo Image (Image by Author)

```
doggo.shape
```

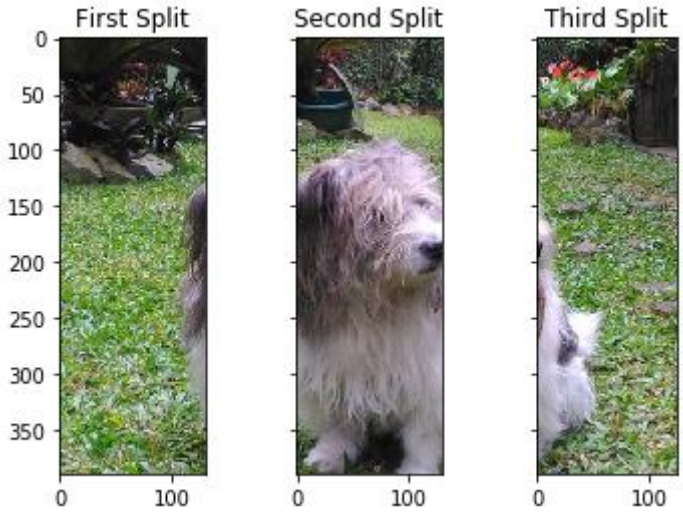
```
(390, 385, 3)
```

Size of Doggo Image

Checking for the size of the image, we see that it is a **390 x 385 x 3** matrix. Remember that using

Python we can slice this matrix and represent each section as its own image box.

```
fig, ax = plt.subplots(1, 3,
                       figsize=(6,4),
                       sharey= True)

ax[0].imshow(doggo[:, 0:130])
ax[0].set_title('First Split')

ax[1].imshow(doggo[:, 130:260])
ax[1].set_title('Second Split')

ax[2].imshow(doggo[:, 260:390])
ax[2].set_title('Third Split');
```



Split Doggo

Remember that although you can do this for all images, each image will have different dimensions. Essentially what I did was find the dimensions that

would split my image into 3 equal parts. Depending on the dimensions of your image, you will have to specify different figures.

Below are the specifications for the dog's face.

```
imshow(doggo[95:250, 130:275]);
```

Face Doggo

Now lets have some fun!

Remember that previously we stated that an image can be represented by a 3 dimensional matrix with each list specifying the color mix.

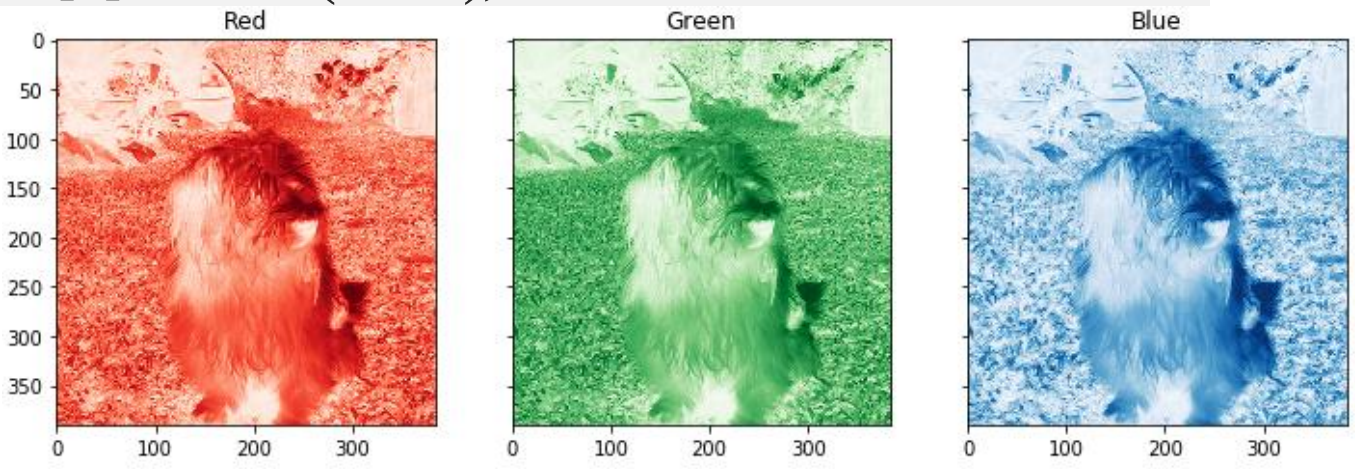
```
Array([[[ 33,  35,  30],
        [ 32,  35,  28],
        [ 33,  36,  29],
        ...,
        [ 35,  33,  35],
        [ 40,  38,  40],
        [ 43,  42,  41]],

       [[ 34,  36,  30],
        [ 33,  36,  29],
        [ 32,  35,  28],
        ...,
        [ 34,  32,  35],
        [ 37,  35,  36],
        [ 44,  43,  42]],

       [[ 34,  36,  30],
        [ 33,  36,  29],
        [ 34,  37,  30],
        ...,
        [ 40,  38,  41],
        [ 37,  35,  36],
        [ 40,  38,  38]],

       ...,

       [[149, 162, 110],
        [153, 166, 117],
        [156, 169, 122],
        ...,
        [ 99, 110,  71],
        [102, 111,  71],
        [ 85,  93,  58]],
```

Matrix Doggo

Each row represents the specific pixel color, we can actually call each of the numbers individually. The numbers can then give us a representation of the image's **Red**, **Green**, and **Blue** components.

```
fig, ax = plt.subplots(1, 3, figsize=(12,4), sharey = True)
ax[0].imshow(doggo[:,:,0], cmap='Reds')
ax[0].set_title('Red')ax[1].imshow(doggo[:,:,1],
```

```
cmap='Greens')
ax[1].set_title('Green')ax[2].imshow(doggo[:,:,2],
cmap='Blues')
ax[2].set_title('Blue');
```

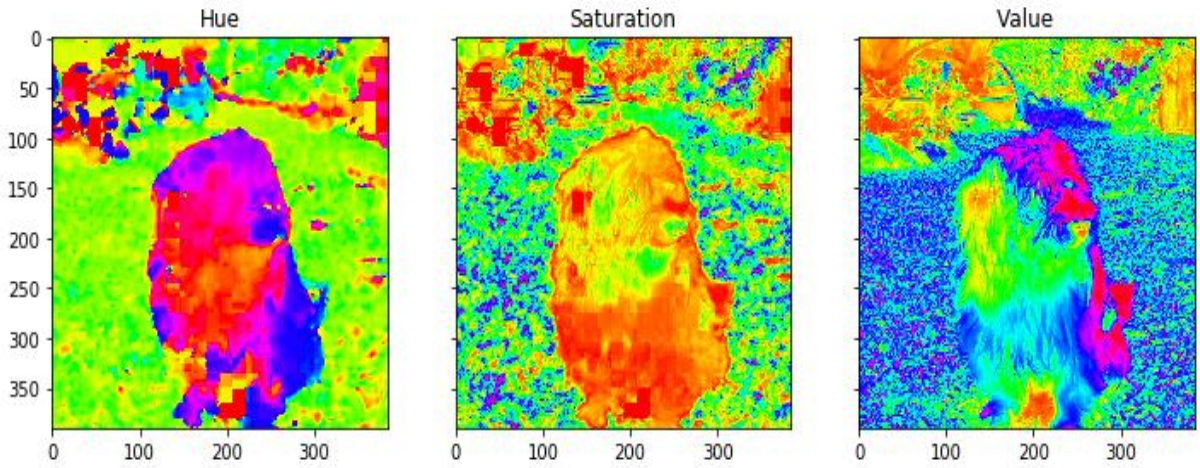


RGB Doggo

Additionally, we can convert the image from RGB (Red, Green, Blue) to HSV (Hue, Saturation, Value).

This will give us the aforementioned figures for Hue, Saturation, and Value.

```
doggo_hsv = rgb2hsv(doggo)fig, ax =
plt.subplots(1, 3, figsize=(12,4), sharey = True)
ax[0].imshow(doggo_hsv[:,:,0], cmap='hsv')
ax[0].set_title('Hue')ax[1].imshow(doggo_hsv[:,:,1],
cmap='gray')
ax[1].set_title('Saturation')ax[2].imshow(doggo_hsv
[:,:,2], cmap='gray')
ax[2].set_title('Value');
```

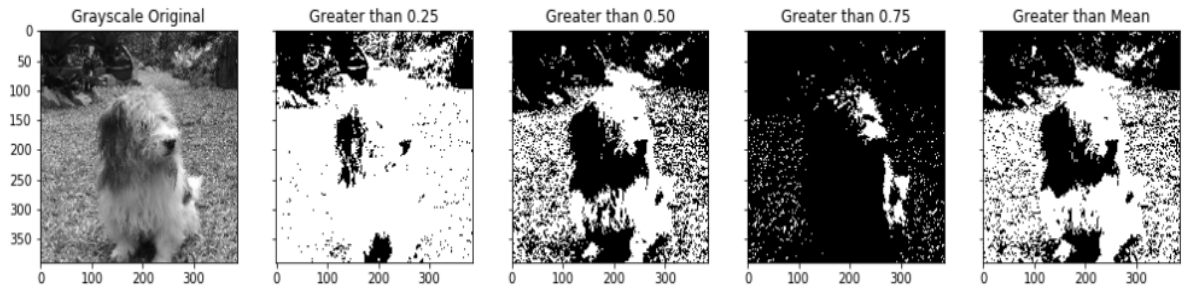


HSV Doggo

Lastly, we can also convert the image matrix to grayscale. By we transform the image to a simple **2x2** matrix. This allows us to easily filter our image based on each pixel's relation to a specified value.

```
doggo_gray = rgb2gray(doggo)fig, ax =
plt.subplots(1, 5, figsize=(17,6), sharey = True)
ax[0].imshow(doggo_gray, cmap = 'gray')
ax[0].set_title('Grayscale
Original')ax[1].imshow(img_as_uint(doggo_gray >
0.25),
            cmap = 'gray')
ax[1].set_title('Greater than
0.25')ax[2].imshow(img_as_uint(doggo_gray >
0.50),
            cmap = 'gray')
ax[2].set_title('Greater than
0.50');ax[3].imshow(img_as_uint(doggo_gray >
```

```
0.75),
            cmap = 'gray')
ax[3].set_title('Greater than
0.75');ax[4].imshow(img_as_uint(doggo_gray >
np.mean(doggo_gray)),
            cmap = 'gray')
ax[4].set_title('Greater than Mean');
```



Binary Doggo

That should give you a basic understanding of how to load and manipulate images using Python. To get a better understanding of image processing, it would be best if you could play around with the codes. I’d also encourage you to start by processing images of topics you actually like. In my case nature has always been a great source of inspiration.