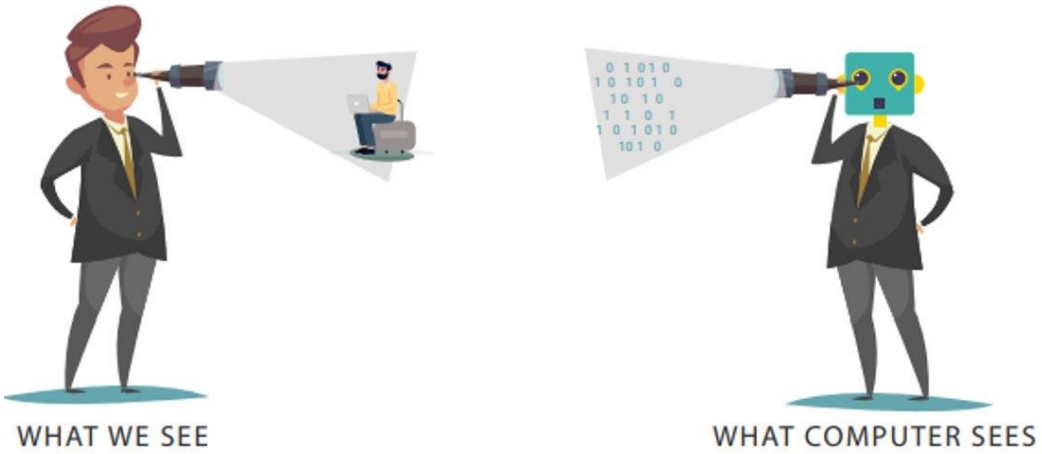
Image processing

Topics Covered

1. Introduction To Image Processing
2. Getting Started with Images
3. Image Processing with OpenCV
4. Image Acquisition and Manipulation Using OpenCV
5. Video Processing
6. Summary

Introduction to Image Processing

Humans have the sense of vision to observe the world using colors and patterns. But computers cannot understand these colors and patterns. Computers see and interpret digital image by breaking them into a grid of small individual units called **pixels**.



Computer Vision

Computer vision is the field of computer science that focuses on creating digital systems that can process, analyze, and make sense of visual data (images or videos) in the same way that humans do. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it.

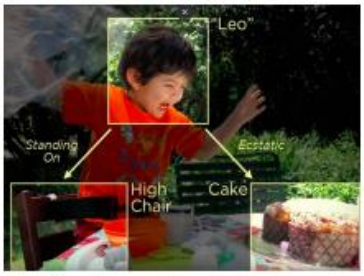
Image Processing

Each image is a three-dimensional arrangement of small pixels units. Hence computers can understand an image as a matrix of numbers. Using this matrix of numbers, computers can understand and interpret the image.

Image Processing is the field of studying and analysing images. The objective is to identify different image features, such as lines, edges, colors and slowly move to advanced topics such as detecting faces and

objects**.** Image Processing is a very active area of research and has huge applications **in** medicine, banking, aerospace and even self-driving cars**.**

Goal Of Computer Vision



1. Naming objects
2. Identifying people
3. Inferring 3D geometry of things
4. Emotions
5. Actions and intentions

Getting Started with Images

- Images are composed of pixels
- all the computers, software are programmed to process pixels in 8 bit size

the pixels values can range from 0000 0000 to 1111 1111 i.e. from 0 to 255

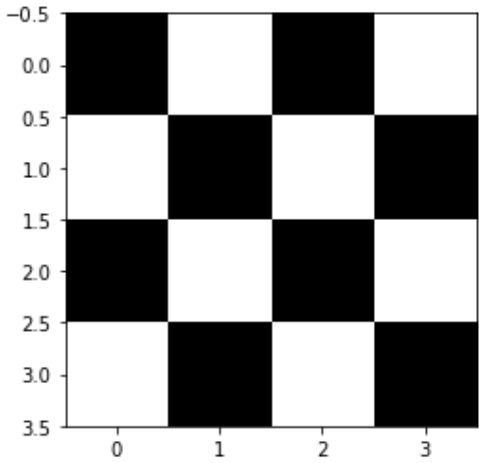
There are primarily two types of images

- Grayscale images - each pixel has one value ranging from 0 to 255
- Color image - each pixel has 3 values ranging from 0 to 255, each value is for - R, G, B

Lets load and manipulate image using numpy

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.array([[0,255,0,255],
              [255,0,255,0],
              [0,255,0,255],
              [255,0,255,0]])
plt.imshow(x,cmap='gray')
plt.show()
```



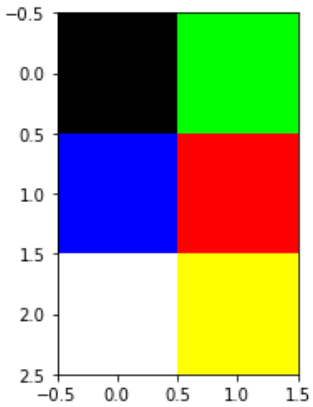
Here we have produced image having Black and white colour having shape of image is (4,4)

Understanding colour images

Each pixel consists of 3 values

- [0,0,0] = Black color
- [255,0,0] = Red Color
- [0,255,0] = Green Color
- [0,0,255] = Blue Color
- [255,255,255] = White color

```
y = np.array([[[0,0,0],[0,255,0]],
              [[0,0,255],[255,0,0]],
              [[255,255,255],[255,255,0]]])

plt.imshow(y)
plt.show()
```



Loading The image

```
## Loading an image from disk
img = plt.imread(r"C:\Users\anshu\Pictures\dog.jpg")
img.shape
```

```
(220, 353, 3)
```

```
%matplotlib inline
plt.imshow(img)
plt.show()
```

Image Processing with OpenCV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

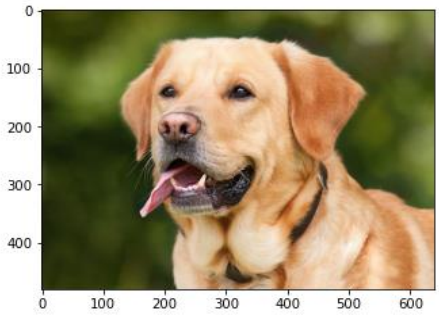
There are thousands of functions available in OpenCV. These simple techniques are used to shape our images in our required format. As we know an image is a combination of pixels, for a color image we have three channels with pixels ranging from 0 to 225, and for black & white-colored images has only one change ranging from 0 to 1.

OpenCV can be directly downloaded and installed with the use of pip (package manager). To install OpenCV, just go to the command-line and type the following command

```
pip install opencv-python.
```

Loading the image with opencv

```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('dog.jpg') # this is read in BGR format
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # this converts it into RGB

plt.imshow(rgb_img)
plt.show()
```



Note – OpenCV arranges the channels in BGR order. So the 0th value will correspond to Blue pixel and not Red.

Shape of Image – Pixel Sizes

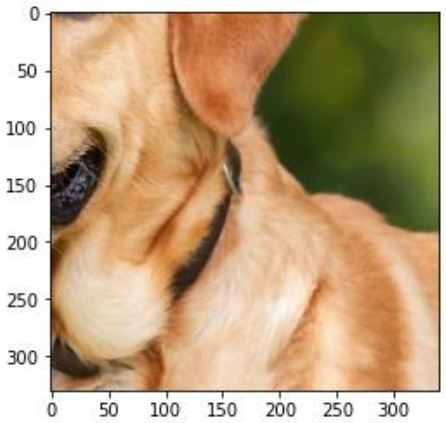
```
# Importing the OpenCV library
import cv2
# Reading the image using imread() function
image = cv2.imread('dog.jpg')

# Extracting the height and width of an image
h, w = image.shape[:2]
# Displaying the height and width
print("Height = {},  Width = {}".format(h, w))
```

```
Height = 480,  Width = 640
```

Image Acquisition and manipulation using OpenCV

Calculate region of interest.

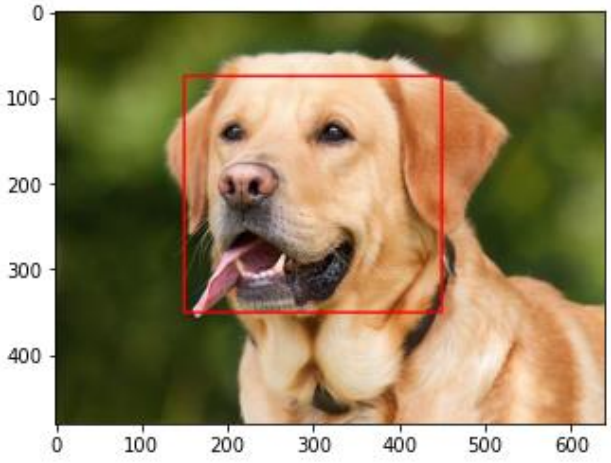
```
# We will calculate the region of interest
# by slicing the pixels of the image
image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
roi = image[150 : 480, 300 : 640]
plt.imshow(roi)
plt.show()
```



Creating the rectangle over image

Here we have passed starting positions and ending positions of rectangle in `cv2.rectangle` function.

```
# Using the rectangle() function to create a rectangle.
rectangle = cv2.rectangle(image, (150, 75),
                          (450, 350), (255, 0, 0), 2)
plt.imshow(rectangle)
plt.show()
```

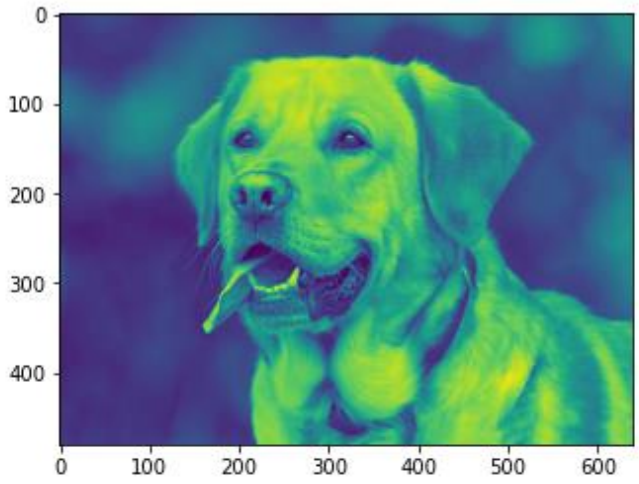


Converting a Colour Image to a Grayscale Image

This is the basic technique that is used to convert the color image into grey shades. A color image consists of 3 channel depth while using Gray scaling it reduces the depth of the image to 1 channel. It reduces model complexity. For many applications like edge detection, photo sketch, cartooning image we use grayscale converted images.

```
grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(grayImage)
```

```
<matplotlib.image.AxesImage at 0x1cbcde1a400>
```

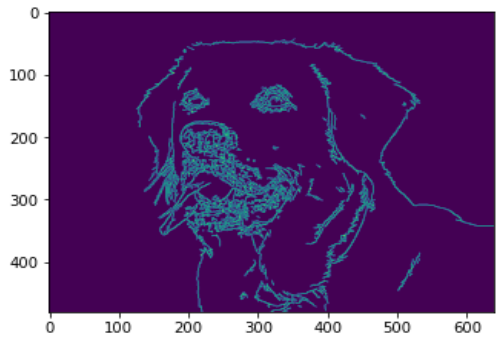


Edge detection

In edge detection, it gives an outline of the image by extracting edges from the image. We are using a canny filter to perform this task. Based on the threshold values, a canny filter detects the edges. The higher thresholds give cleaner images compared to lower thresholds gives a clumsy output.

```
img = cv2.imread('dog.jpg',0)

edge_det = cv2.Canny(img,100,200) # Using Canny filter for edge detection

plt.imshow(edge_det)
```

```
<matplotlib.image.AxesImage at 0x1cbcee39d90>
```



Here we had given a threshold value 100,200 and you can try with different threshold values. This edge detection is used for segmentation, restoration, picture enhancement, pattern recognition.

Smoothing Techniques

The image smoothing technique is performed using a filter. By convolving the image, it reduces the noise in the image by adding a blurring effect on the edges. There are different types of smoothing techniques we perform depending on the input image.

Average using cv2.blur()

In this technique, depending on the size of the filter convolution is performed but it averages the pixels under the filter and assigns the value to the centre pixel under the filter.

```
img = cv2.imread('dog.jpg')
image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

blur = cv2.blur(image,(20,20))

plt.imshow(blur)
#plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x1cbcdcfd190>
```

Median Blur using cv2.medianBlur()

In this technique, it calculates the median of the pixels under the filter and it replaces the center value under the filter with the median value, positive odd integer to be assigned as filter size to perform the median blur technique.

Comparing smoothing techniques averaging and median blur gives similar results because one calculates on averaging pixels and another on calculating the medians under the filter.

Gaussian Blur using cv2.GaussianBlur()

In this technique, there is another parameter called sigma x and sigma y. It performs a blurring of an image by using parameters sigma x & sigma y. If it is set to zero then it calculates based on the size of its kernel.

Morphological Techniques

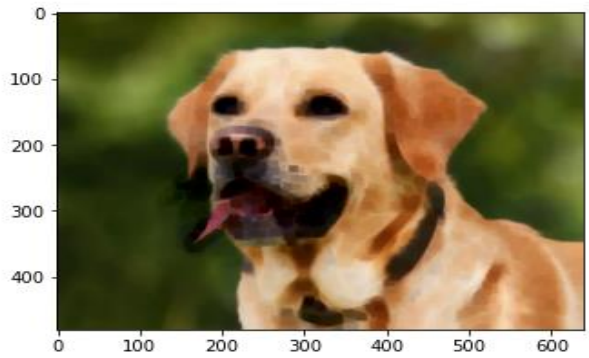
These are the techniques used to manipulate the image sizes, these techniques are used only after converting the image into grayscale. These techniques are used for removing noise, correcting the imperfections in the data, and to make clear images. Erosion and Dilation are mostly used in morphological techniques.

Erosion

This technique removes the boundary pixels from the input by just passing the filter on the image. Depending on the size of the kernel, it removes the boundary pixels from the input image. It performs similar to soil erosion so they named this technique as erosion.

```
img = cv2.imread('dog.jpg')
image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

kernel = np.ones((10,10),np.uint8)

img_erosion = cv2.erode(image, kernel, iterations=1)

plt.imshow(img_erosion)
```

```
<matplotlib.image.AxesImage at 0x1cbd000c190>
```

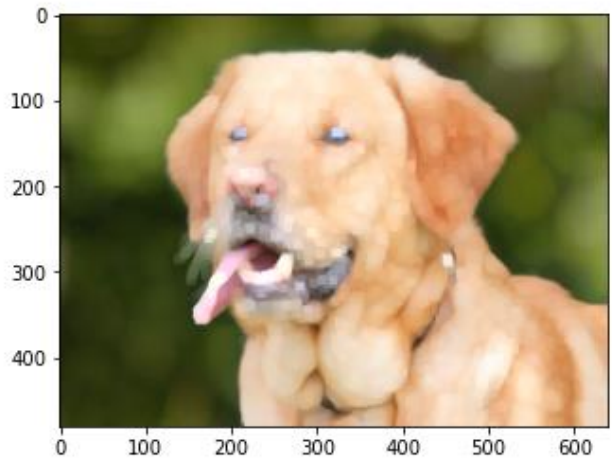


Dilation

The above technique removes the boundary pixels but in Dilation. It adds the additional pixels to the input. It is used when the pixels are missing in the image. In general practices, we apply erosion to shrink the image to remove noises, and then by applying the dilation, there will be no loss of pixels.

```
img = cv2.imread('dog.jpg')
image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

kernel = np.ones((10,10),np.uint8)

img_dilation = cv2.dilate(image, kernel, iterations=1)

plt.imshow(img_dilation)
```

```
<matplotlib.image.AxesImage at 0x1cbd19cecd0>
```



Video Processing

Processing a video means, performing operations on the video frame by frame. Frames are nothing but just the particular instance of the video in a single point of time. We may have multiple frames even in a single second. Frames can be treated as similar to an image.

So, whatever operations we can perform on images can be performed on frames as well. Let us see some of the operations with examples.

let's take a look at video processing using an OpenCV and Python:

First of all, we are creating a cv2.VideoCapture object, cv2.VideoCapture is a class for video capturing from video files, image sequences, or cameras.

```
import cv2 #source might be provided as video filename or integer number for camera capture

cap = cv2.VideoCapture(source)
```

Then, in order to play video, we create a non-trivial cycle:

```
while True:
… ret, frame = cap.read()
```

It's interesting that it's an unbounded cycle.

```
while True:
… ret, frame = cap.read()
… cv2.imshow('window name', frame)
```

The `cv2.imshow` method displays an image in the specified window. We specify a window name as a first argument, and the frame we would like to display as a second. And now, we need to somehow break an infinite cycle:

```
while True:
… ret, frame = cap.read()
… cv2.imshow('window name', frame)
… if cv2.waitKey(1) & 0xFF == ord('q'):
… break
… cap.release()
… cv2.destroyAllWindows()
```

`cv2.waitKey()` is a required building block for OpenCV video processing. `waitKey` is a method which displays the frame for specified milliseconds. The '0xFF == ord('q')' inside the 'if' statement is a special syntax to provide the 'while' loop break, by a keyboard key pressing event.

`cap.release()` and `cv2.destroyAllWindows()` are the methods to close video files or the capturing device, and destroy the window, which was created by the `imshow` method.

Summary

- Computer vision is the field of computer science that focuses on creating digital systems that can process, analyze, and make sense of visual data (images or videos) in the same way that humans
- Image Processing is the field of studying and analyzing images. The objective is to identify different image features, such as lines, edges, colors and slowly move to advanced topics such as detecting faces and objects.
- Erosion technique removes the boundary pixels from the input by just passing the filter on the image.
- Dilation adds the additional pixels to the input. It is used when the pixels are missing in the image.
- Processing a video means, performing operations on the video frame by frame. Frames are nothing but just the particular instance of the video in a single point of time.