Computer Vison

What Is Computer Vision?

To simplify the answer to this – Let us consider a scenario.

We all use Facebook, correct? Let us say you and your friends went on a vacation and you clicked a lot of pictures and you want to upload them on Facebook and you did. But now, wouldn't it take so much time just to find your friends faces and tag them in each and every picture? Well, Facebook is intelligent enough to actually tag people for you.

So, how do you think the auto tag feature works? In simple terms, it works on computer vision.

Computer Vision is an interdisciplinary field that deals with how computers can be made to gain a high-level understanding from digital images or videos.

The idea here is to automate tasks that the human visual systems can do. So, a computer should be able to recognize objects such as that of a face of a human being or a lamppost or even a statue.

How Does A Computer Read An Image?

Consider the below image:

OpenCV Python Tutorial - EdurekaWe can figure out that it is an image of the New York Skyline. But, can a computer find this out all on its own? The answer is no!

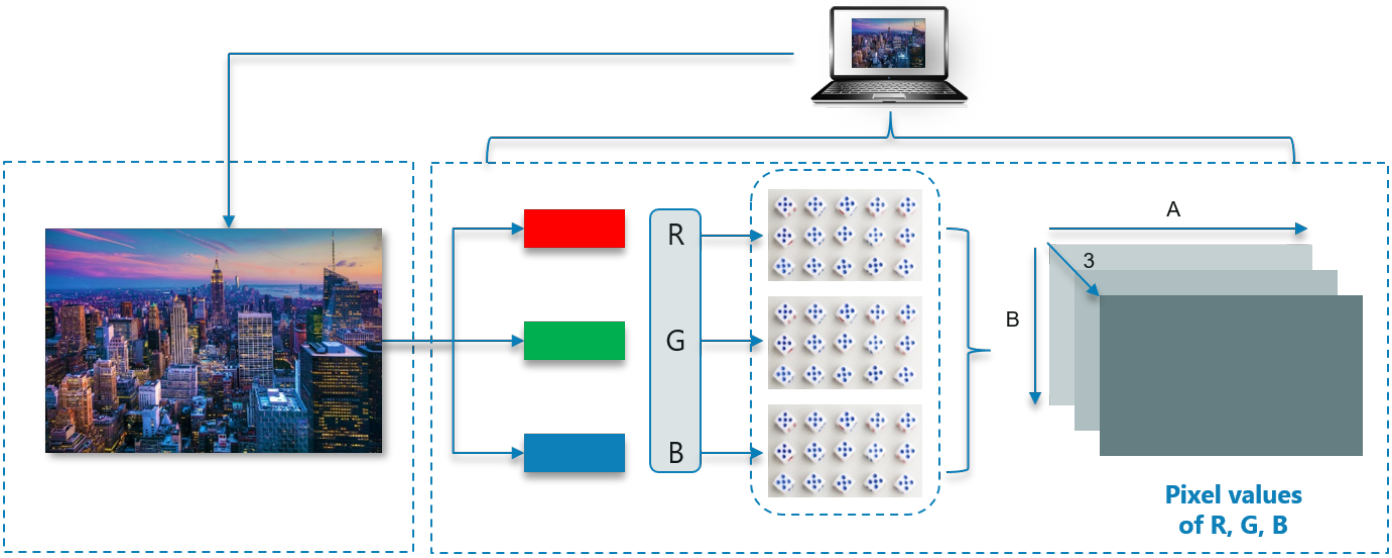
The computer reads any image as a range of values between 0 and 255.

For any color image, there are 3 primary channels – Red, green and blue. How it works is pretty simple.

A matrix is formed for every primary color and later these matrices combine to provide a Pixel value for the individual R, G, B colors.

Each element of the matrices provide data pertaining to the intensity of brightness of the pixel.

Considering the following image:



As shown, the size of the image here can be calculated as B x A x 3.

Note: For a black-white image, there is only one single channel.

Next up on this OpenCV Python Tutorial blog, let us look at what OpenCV actually is.

What Is OpenCV?

OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage.

OpenCV supports a wide variety of programming languages such as C++, Python, Java etc. Support for multiple platforms including Windows, Linux, and MacOS.

OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures gets converted to/from NumPy arrays.

This makes it easier to integrate it with other libraries which use NumPy. For example, libraries such as SciPy and Matplotlib.

Next up on this OpenCV Python Tutorial blog, let us look at some of the basic operations that we can perform with OpenCV.

Read and Display Image

In Computer Vision applications, images are an integral part of the development process. Often there would be a need to read images and display them if required.

In this tutorial, we will learn how to read and display an image using OpenCV.

To read and display image using OpenCV Python, you could use cv2.imread() for reading image to a variable and cv2.imshow() to display the image in a separate window.

In [2]:

```
# import OpenCV first

import cv2

#The syntax of imread() function is

cv2.imread(/complete/path/to/image,flag)
```

First argument is complete path to the image along with the extension.

Second argument is an optional flag which could be any one of the following.

- cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.
- cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode
- cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel Returns numpy array, containing the pixel values. For colored images, each pixel is represented as an array containing Red, Green and Blue channels.

Note that the default flag is cv2.IMREAD_COLOR. Hence even if read a png image with transparency, the transparency channel is neglected.

In [ ]:

```
#The syntax of imshow() function is
cv2.imshow(window_name, image)
```

First argument is name of the window that is displayed when image is shown in.

Second argument is the image to be shown in the window.

In [ ]:

```
# Full implementation with any image
import cv2

img = cv2.imread('/home/img/python.png')

cv2.imshow('sample image',img)

cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
```

Save Image

In this tutorial, we will learn how to save image data from ndarray to a file, in OpenCV Python using imwrite() function, with an example.

While working with images in Image Processing applications, it is quite often that you need to store intermediate results of image transformations or save the final resulting image. When working with OpenCV Python, images are stored in numpy ndarray. To save an image to the local file system, use cv2.imwrite() function of opencv

python library.

In [ ]:

```
#The syntax of cv2.imwrite() function is

cv2.imwrite('/path/to/destination/image.png',image)
```

where

- First Argument is Path to the destination on file system, where image is ought to be saved.
- Second Argument is ndarray containing image
- Returns True is returned if the image is written to file system, else False.

In [ ]:

```
#In this example, we will read an image, then transform it to grey image and save this im

import cv2

# read image as grey scale
grey_img = cv2.imread('/home/img/python.png', cv2.IMREAD_GRAYSCALE)

# save image
status = cv2.imwrite('/home/img/python_grey.png',grey_img)

print("Image written to file-system : ",status)
```

Run the above python script.

```
Image written to file-system :  True
```

cv2.imwrite() returned true which means the file has been successfully written to the path specified. Reading the return value of imwrite() is very important as sometimes there could be multiple reasons that fail the disk write operation and resulting in the image not written to disk.

Get Image Size

In Image Processing applications, it is often necessary to know the size of an image that is loaded or transformed through various stages.

In this OpenCV Tutorial, we will learn how to get image size in OpenCV Python with an example.

When working with OpenCV Python, images are stored in numpy ndarray. To get the image shape or size, use ndarray.shape to get the dimensions of the image. Then, you can use index on the dimensions variable to get width, height and number of channels for each pixel.

In [ ]:

```
#In the following code snippet, we have read an image to img ndarray. And then we used nd

img = cv2.imread('/path/to/image.png')
dimensions = img.shape
```

In this example, we have read an image and used ndarray.shape to get the dimension.

We can access height, width and number of channels from img.shape: Height is at index 0, Width is at index 1; and number of channels at index 2.

In [ ]:

```
import cv2

# read image
img = cv2.imread('/home/img/python.png', cv2.IMREAD_UNCHANGED)

# get dimensions of image
dimensions = img.shape

# height, width, number of channels in image
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2]

print('Image Dimension    : ',dimensions)
print('Image Height       : ',height)
print('Image Width        : ',width)
print('Number of Channels : ',channels)
```

```
Image Dimension    :  (149, 200, 4)
Image Height       :  149
Image Width        :  200
Number of Channels :  4
```

img.shape returns (Height, Width, Number of Channels)

where

- Height represents the number of pixel rows in the image or the number of pixels in each column of the image array.
- Width represents the number of pixel columns in the image or the number of pixels in each row of the image array.

- Number of Channels represents the number of components used to represent each pixel.

In the above example, Number of Channels = 4 represent Alpha, Red, Green and Blue channels.

Resize image

Resizing an image means changing the dimensions of it, be it width alone, height alone or changing both of them. Also, the aspect ratio of the original image could be preserved in the resized image. To resize an image, OpenCV provides cv2.resize() function.

In this tutorial, we shall the syntax of cv2.resize and get hands-on with examples provided for most of the scenarios encountered in regular usage.

In [ ]:

```
#The syntax of resize function in OpenCV is

cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

where

- src ------->[required] source/input image
- dsize --->[required] desired size for the output image
- fx -------->[optional] scale factor along the horizontal axis
- fy -------->[optional] scale factor along the vertical axis
- interpolation [optional] flag that takes one of the following methods.

  - INTER_NEAREST – a nearest-neighbor interpolation INTER_LINEAR – a bilinear interpolation (used by default)
  - INTER_AREA – resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
  - INTER_CUBIC – a bicubic interpolation over 4×4 pixel neighborhood INTER_LANCZOS4 – a Lanczos interpolation over 8×8 pixel neighborhood

Resizing an image can be done in many ways. We will look into examples demonstrating the following resize operations.

1. Preserve Aspect Ratio (height to width ratio of image is preserved)
   - Downscale (Decrease the size of the image)
   - Upscale (Increase the size of the image)
2. Do not preserve Aspect Ratio
   - Resize only the width (Increase or decrease the width of the image keeping height unchanged)
   - Resize only the height (Increase or decrease the height of the image keeping width unchanged)
   - Resize to specific width and height

Following is the original image with dimensions (149,200,4)(height, width, number of channels) on which we shall experiment on:

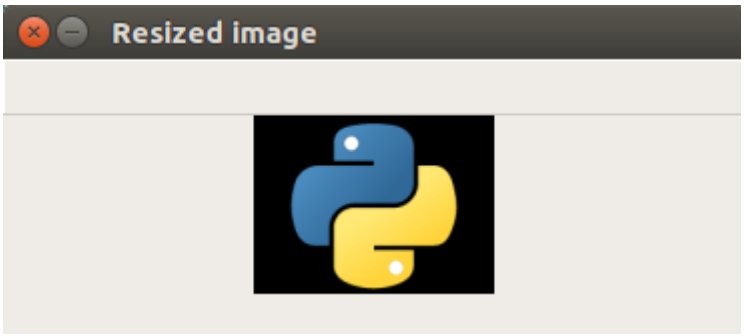
In the following example, scale_percent value holds the percentage by which image has to be scaled. Providing a value <100 downscales the image provided. We will use this scale_percent value along with original image's dimensions to calculate the width and height of output image.

In [ ]:

```
import cv2

img = cv2.imread('/home/img/python.png', cv2.IMREAD_UNCHANGED)

print('Original Dimensions : ',img.shape)

scale_percent = 60 # percent of original size
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)

# resize image
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

print('Resized Dimensions : ',resized.shape)

cv2.imshow("Resized image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Original Dimensions :  (149, 200, 4)
Resized Dimensions :  (89, 120, 4)
```

The original image with dimensions [149 x 200 x 4] has been resized to [89, 120, 4] using resize() function.



Edge Detection

Edge Detection is an image processing technique to find boundaries of objects in the image.

In this tutorial, we shall learn to find edges of focused objects in an image using Canny Edge Detection Technique.

In [ ]:

```
The syntax of OpenCV Canny Edge Detection function is

edges = cv2.Canny('/path/to/img', minVal, maxVal, apertureSize, L2gradient)
```

where

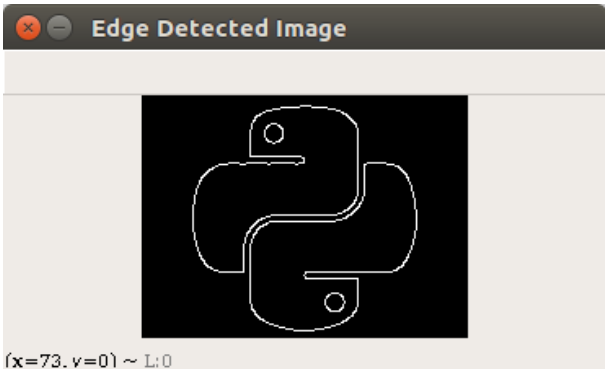
- /path/to/img ---->(Mandatory) File Path of the image
- minVal ---------->(Mandatory) Minimum intensity gradient
- maxVal ---------->(Mandatory) Maximum intensity gradient
- apertureSize ----->(Optional)
- L2gradient -------->(Optional) (Default Value : false) If true, Canny() uses a much more computationally expensive equation to detect edges, which provides more accuracy at the cost of resources.

In [ ]:

```
#In this example, we python.png (an RGB image) as a GREY scale image. Then Canny() functi

import cv2

img = cv2.imread('/home/img/python.png')
edges = cv2.Canny(img,100,200)

cv2.imshow("Edge Detected Image", edges)

cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
```

Input Image

Output Image


Edge Detected Image
(x=73, y=0) ~ L:0