Face Mask Detection using Python

The COVID-19 pandemic, also known as the coronavirus pandemic is the greatest humanitarian challenge faced by the world. The pandemic has spread widely, bringing the world to a halt and the number of cases continue to rise. Governments all around the world are working to slow down if not completely stopped the spread of this pandemic. They have implemented many protocols and one very crucial one is wearing of mask to avoid infection till vaccinations become accessible to all. Computer Vision in the way of Mask Detection is a reviving factor to get our lives back on track. Real Time mask detection can solve the monitoring issues in geographies with high population.

Steps to perform Image Processing: -

- ✓ Load images using python or any other programming
- ✓ Convert images into array
- ✓ After loading images, we can perform any algorithm on array we have.

All the above steps can be performed using **OpenCV.**

OpenCV:

OpenCV (Open Source Computer Vision Library**)-Python** is a library of Python bindings designed to solve computer vision problems. Computer vision is the field of computer science that deals with how computers can gain high-level understanding from digital images or videos. In OpenCV all the images are converted to NumPy arrays.

Some Features of OpenCV:

a) Face Detection
b) Geometric Transformation
c) Smoothing Images
d) Canny Edge Detection
e) Background Removals
f) Image Segmentation
g) Image Thresholding

Getting Started with OpenCV-Python:

i. To use this library first of all install it.
ii. After successfully installing **OpenCV-Python** library, import it**.**
For importing this library following syntax must be followed: -

Synatx:

```
import cv2
```

iii. Once this library is imported successfully then you can perform your task easily.

NumPy: -

NumPy is a Python library used for working with arrays. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Getting Started with NumPy:

i. To use this library first of all install it.

ii. After successfully installing **NumPy** library, import it**.**
For importing this library following syntax must be followed: -

```
Syntax:
    import numpy
```

Now NumPy is imported and ready to use.

iii. NumPy is usually imported under the np alias.

```
Syntax:
    import numpy as np
```

Haar Cascades for Object Detection

Object Detection is a computer technology related to computer vision, image processing and deep learning that deals with detecting instances of objects in images and videos.

What is Haar Cascade?

It is an Object Detection Algorithm used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper "Rapid Object Detection using a Boosted **Cascade** of Simple Features" published in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. OpenCV already contains many pre-trained classifiers (cascade function) for face, eyes, smile etc. Those XML files are stored in OpenCV Github repository.

i. **Positive images –** These images contain the images which we want our cascade function to identify.
ii. **Negative Images –** Images of everything else, which do not contain the object we want to detect.

Face Detection

1. An image in which the face is to be detected.
2. Create a cascade classifier. It will contain the features of the face.
3. OpenCV will read the image and the features file and will convert the image into NumPy arrays.
4. OpenCV will search for the row and column values of the face NumPy ndarray.
5. After that we will display the image with the rectangular face box.

cv2.CascadeClassifier(): - This loads the Haar Cascade file.

Syntax:

```
cv2.CascadeClassifier('Haar Cascade_filename')
```

```
haar_data = cv2.CascadeClassifier('data.xml')
```

Create a CascadeClassifier Object

Path to the xml file which contains the face features

cv2.VideoCapture( ):- It is used to create a video capture object which is helpful to capture videos through webcam and then desired operations on that video can be performed. Its argument is the device index. A device index is just the number to specify which camera. Normally one camera will be connected. So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on.

```
Syntax:
    cv2.VideoCapture(device index)
```

After this, we can start reading a Video from the camera frame by frame. We do this by calling the read method on the **VideoCapture** object.

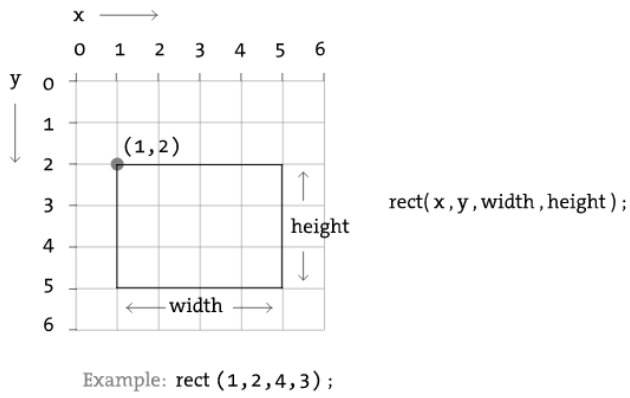
```
Sytax:
    VideoCapture_Object.read()
```

This method takes no arguments and returns a tuple. The first returned value is a Boolean indicating if the camera was read correctly (**True**) or not (**False**). The second returned value is the image or the video captured.

After the camera is used, it needs to be released. This is done using the release function.

```
Sytax:
    VideoCapture_Object.release()
```

detectMultiScale( ):- It is used to detect the objects of different sizes in the input image. This function will return a rectangle with coordinates(x, y, width, height) around the detected face.



cv2.rectangle():- cv2.rectangle() method is used to draw a rectangle on any image.

```
Syntax:

cv2.rectangle(image, start_point, end_point, color, thickness)

Parameters:

image: It is the image on which rectangle is to be drawn.

start_point: It is the starting coordinates of rectangle. The coordinates are
represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

end_point: It is the ending coordinates of rectangle. The coordinates are represented
as tuples of two values i.e. (X coordinate value, Y coordinate value).

color: It is the color of border line of rectangle to be drawn. For BGR, we pass a
tuple. eg: (255, 0, 0) for blue color.

thickness: It is the thickness of the rectangle border line in px. Thickness of -1 px
will fill the rectangle shape by the specified color.

Return Value: It returns an image.
```

cv2.resize(): To **resize** an image, **OpenCV** provides **cv2**. **resize**() function.

```
Syntax:
    cv2.resize(src, dsize)
```

Where,
src- source/input image
dsize- desired size for the output image

numpy.save():- Save an array to a binary file in NumPy .npy format.

```
Syntax:-
        numpy.save('file_name.npy', arr)
```

Where,

file_name is the name of the file and **arr** is the data to be saved.

numpy.load:- This function return the input array from a disk file with npy extension(.npy).

```
Syntax:-
        numpy.load('file_name.npy')
```

numpy.reshape:- Gives a new shape to an array without changing its data.

```
Syntax:-
        numpy.reshape()
```

numpy.r_ :- This is used to concatenate any number of array slices along row (first) axis.

```
Syntax:-
        numpy.r_[ ]
```

numpy.zeros( ): - numpy.zeros() or np.zeros is used to create a matrix full of zeroes.

```
Syntax:-
        numpy.zeros()
```

cv2.imshow( ) :- cv2.imshow() method is used to display an image in a window. The window automatically fits to the image size.

```
Synatx:

cv2.imshow('window_name',image)
```

cv2.destroyAllWindows():- This simply destroys all the windows we created.

```
Synatx:

cv2.destroyAllWindows()
```

cv2.waitKey():- It is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event.

```
Synatx:

cv2.waitKey()
```

Basic concepts of Machine Learning: -

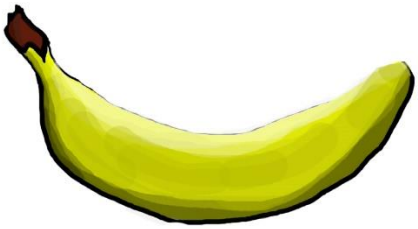
- Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.

- Scikit-learn (formerly scikits.learn and also known as sklearn) is a machine learning library for the Python programming language.

Supervised learning

- Supervised learning, also known as supervised machine learning, as the name indicates, has the presence of a supervisor as a teacher. Basically, supervised learning is when we teach or train the machine using data that is well labeled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labeled data.
- **For instance**, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

  - If the shape of the object is rounded and has a depression at the top, is red in color, then it will be labeled as –**Apple**.
  - If the shape of the object is a long curving cylinder having Green-Yellow color, then it will be labeled as –**Banana**.
- Now suppose after training the data, you have given a new separate fruit, say Banana from the basket, and asked to identify it.



- Since the machine has already learned the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in the Banana category. Thus, the machine learns the things from training data (basket containing fruits) and then applies the knowledge to test data (new fruit).

Features in machine learning

In Machine Learning feature means property of your training data. Or you can say a column name in your training dataset.

Suppose this is your training dataset

Height	Sex	Age
61.5	M	20
55.5	F	30
64.5	M	41
55.5	F	51
.	.	.
.	.	.
.	.	.
.	.	.

Then here Height, Sex and Age are the features.

Labels in machine learning

The output you get from your model after training it is called a label.

Suppose you fed the above dataset to some algorithm and generates a model to predict gender as Male or Female, In the above model you pass features like age, height etc.

So, after computing, it will return the gender as Male or Female. That's called a **Label.**

Train-Test Split Evaluation

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset. A **model** represents what was learned by a **machine learning** algorithm. The **model** is the "thing" that is saved after running a **machine learning** algorithm on training data.

- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values.

The train-test procedure is appropriate when there is a sufficiently large dataset available.

When to Use the Train-Test Split

The idea of "sufficiently large" is specific to each predictive modeling problem. It means that there is enough data to split the dataset into train and test datasets and each of the train and test datasets are suitable representations of the problem domain. This requires that the original dataset is also a suitable representation of the problem domain.

A suitable representation of the problem domain means that there are enough records to cover all common cases and most uncommon cases in the domain. This might mean combinations of input variables observed in practice. It might require thousands, hundreds of thousands, or millions of examples.

Conversely, the train-test procedure is not appropriate when the dataset available is small. The reason is that when the dataset is split into train and test sets, there will not be enough data in the training dataset for the model to learn an effective mapping of inputs to outputs. There will also not be enough data in the test set to effectively evaluate the model performance. The estimated performance could be overly optimistic (good) or overly pessimistic (bad).

If you have insufficient data, then a suitable alternate model evaluation procedure would be the k-fold cross-validation procedure.

In addition to dataset size, another reason to use the train-test split evaluation procedure is computational efficiency.

Some models are very costly to train, and in that case, repeated evaluation used in other procedures is intractable. An example might be deep neural network models. In this case, the train-test procedure is commonly used.

Alternately, a project may have an efficient model and a vast dataset, although may require an estimate of model performance quickly. Again, the train-test split procedure is approached in this situation.

Samples from the original training dataset are split into the two subsets using random selection. This is to ensure that the train and test datasets are representative of the original dataset.

How to Configure the Train-Test Split

The procedure has one main configuration parameter, which is the size of the train and test sets. This is most commonly expressed as a percentage between 0 and 1 for either the train or test datasets. For example, a training set with the size of 0.67 (67 percent) means that the remainder percentage 0.33 (33 percent) is assigned to the test set.

There is no optimal split percentage.

You must choose a split percentage that meets your project's objectives with considerations that include:

- Computational cost in training the model.
- Computational cost in evaluating the model.
- Training set representativeness.
- Test set representativeness.

Nevertheless, common split percentages include:

- Train: 80%, Test: 20%
- Train: 67%, Test: 33%
- Train: 50%, Test: 50%

Now that we are familiar with the train-test split model evaluation procedure, let's look at how we can use this procedure in Python.

Train-Test Split Procedure in Scikit-Learn

The scikit-learn Python machine learning library provides an implementation of the train-test split evaluation procedure via the train_test_split() function. Firstly, we import the **train_test_split** function from the **model_selection** module of Scikit-Learn.

```
from sklearn.model_selection import train_test_split
```

The function takes a loaded dataset as input and returns the dataset split into two subsets.

```
# split into train test sets
train, test = train_test_split(dataset, ...)
```

Ideally, you can split your original dataset into input (*X*) and output (*y*) columns, then call the function passing both arrays and have them split appropriately into train and test subsets.

```
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, ...)
```

The size of the split can be specified via the "test_size" argument that takes a number of rows (integer) or a percentage (float) of the size of the dataset between 0 and 1.

The latter is the most common, with values used such as 0.33 where 33 percent of the dataset will be allocated to the test set and 67 percent will be allocated to the training set.

```
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

We can demonstrate this using a synthetic classification dataset with 1,000 examples.

The complete example is listed below.

```
# split a dataset into train and test sets
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_blobs(n_samples=1000)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

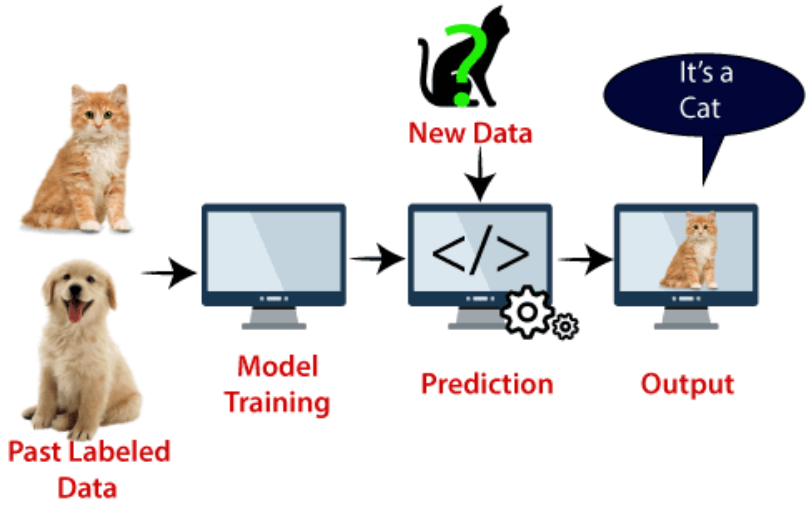
Running the example splits the dataset into train and test sets, then prints the size of the new dataset.

We can see that 670 examples (67 percent) were allocated to the training set and 330 examples (33 percent) were allocated to the test set, as we specified.

```
(670, 2) (330, 2) (670,) (330,)
```

Alternatively, the dataset can be split by specifying the "train_size" argument that can be either a number of rows (integer) or a percentage of the original dataset between 0 and 1, such as 0.67 for 67 percent.

```
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.67)
```



Accuracy score of a model

Scikit-Learn provides a function, **accuracy_score,** which accepts the true value and predicted value as its input to calculate the accuracy score of a model.

Firstly, we import the **accuracy_score** function from the **metrics** module of Scikit-Learn.

```
from sklearn.metrics import accuracy_score
```

Next, we input the **y_test** as true value and **y_pred** as the predicted value to the **accuracy_score** function.

```
accuracy_score(y_test , y_pred)
```

Classification Problem

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories.

From a modeling perspective, classification requires a training dataset with many examples of inputs and outputs from which to learn.

A model will use the training dataset and will calculate how to best map examples of input data to specific class labels. As such, the training dataset must be sufficiently representative of the problem and have many examples of each class label.

For example, spam detection in email service providers can be identified as a classification problem. In this there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email.

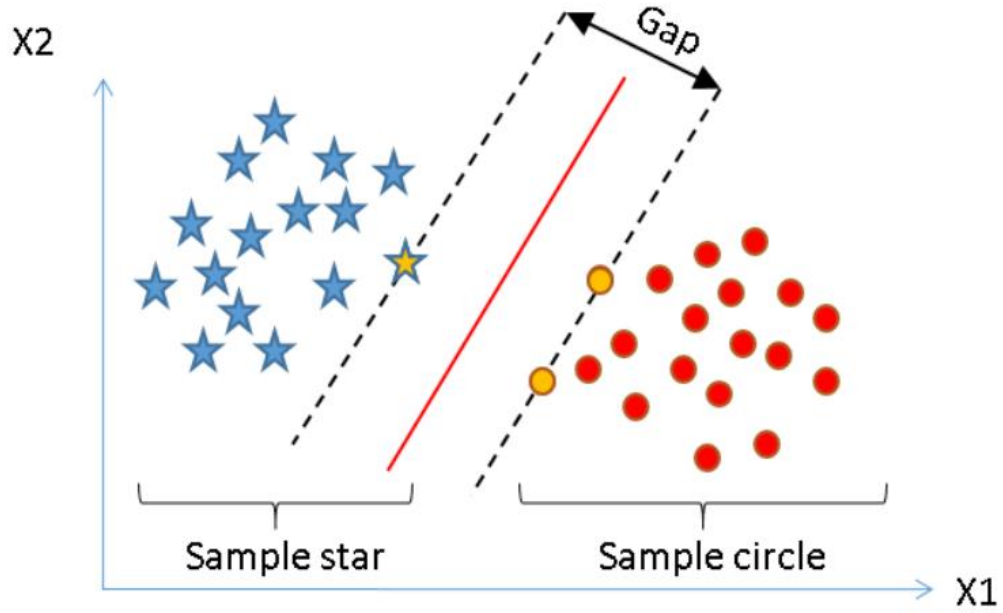
Class labels are often string values, e.g. "*spam*," "*not spam*," and must be mapped to numeric values before being provided to an algorithm for modeling. This is often referred to as label encoding, where a unique integer is assigned to each class label, e.g. "*spam*" = 0, "*no spam*" = 1.

Support Vector Machine or SVM algorithm

Support Vector Machine or SVM algorithm is a simple yet powerful Machine Learning algorithm used for classification problem. It falls under the category of Supervised learning algorithms and uses the concept of Margin to classify between classes.

Given classes X1 and X2, we want to find the decision boundary which separates the 2 classes the best i.e. with minimum error.

SVM does this with a **'Hyperplane'**. Now this hyperplane can be a single line in case of a 2-dimensional data and can be a plane in 3-dimensional one.



Support Vector Machines uses the concept of **'Support Vectors '**, which are the closest points to the hyperplane.

In the above example, the red line denotes our decision boundary that separates the 2 classes (Blue stars and Red circles) and the hyphened lines represent our **'Margin'**, the gap we want between the Support Vectors of both the classes.

The Margin is defined with the help of the Support Vectors (hence the name). In our example, Yellow stars and Yellow circles are the Support Vectors defining the Margin. The better the gap, the better the classifier works. Hence support vectors play an important role in developing the classifier.

Every new data point in test data will be classified according to this Margin. If it lies on the right side of it, it'll be classified as a Red circle otherwise as a Blue star.

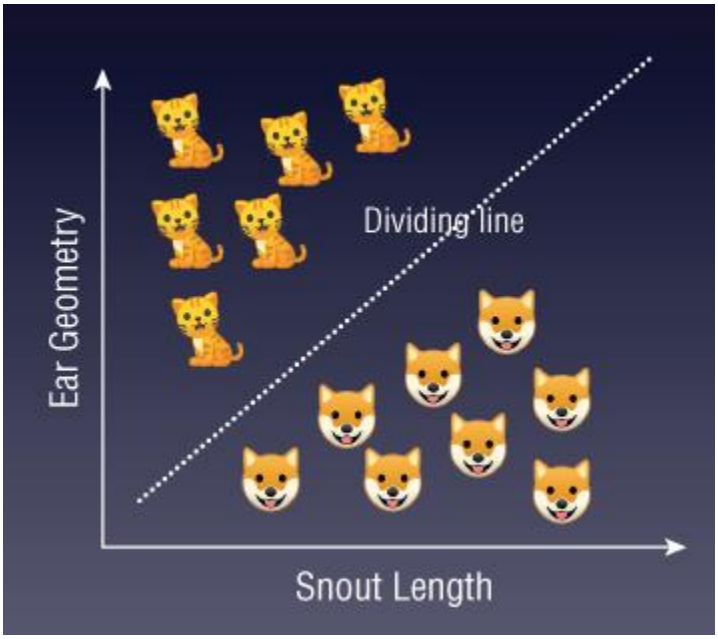


Figure 8.1: Using SVM to separate two classes of animals

Once the line is drawn to separate the classes, you can then use it to predict future data. For example, given the snout length and ear geometry of a new unknown animal, you can now use the dividing line as a classifier to predict if the animal is a dog or a cat.

Generating Model

Let's build support vector machine model. First, import the SVM module and create support vector classifier using SVC() function.

An algorithm that implements classification, especially in a concrete implementation, is known as a classifier.

Then, fit your model on train set using fit() and perform prediction on the test set using predict().

```
#Import svm model
from sklearn.svm import SVC

#Create a svm Classifier
svm = SVC()

#Train the model using the training sets
svm.fit(x_train , y_train)

#Predict the response for test dataset
y_pred = svm.predict(x_test)
```

Principal Component Analysis

- High dimension data is extremely complex to process due to inconsistencies in the features which increase the computation time and make data processing more convoluted.
- Principal components analysis (PCA) is a dimensionality reduction technique that enables you to identify correlation and patterns in a data set so that it can be transformed into a data set of significantly lower dimension without loss of any important information.
- It is very easy to perform PCA using Python's Scikit-Learn library. Firstly, we import the PCA function from the decomposition module of Scikit-Learn. The PCA class is used for this purpose. Firstly, we import the PCA function from the decomposition module of Scikit-Learn. Initialize the PCA class by passing the number of components to the constructor. The most important hyperparameter in PCA class is n_components.

```
# PCA - Principal Component Analysis
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
```

In the code above, we create a **PCA** object named **pca**. We have specified the number of components in the constructor as 3. This will project our original data onto a three-dimensional subspace.

```
pca.fit_transform(x_train)
```

Call the **fit_transform**() method. **fit_transform**() is used on the training data so that we can scale the training data. **fit_transform**() will apply the dimensionality reduction on **x_train.**

cv2.imread( ) :-

cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix. imread() takes the image as input and decodes into a matrix(array). Each index of array represents (red, green, blue) color pixel which ranges from 0 to 255.

Note: The image should be in the working directory or a full path of image should be given.

```
Synatx:

cv2.imread('path_of_image')
```

cv2.putText(): -

cv2.putText() method is used to draw a text string on any image.

```
Syntax:

cv2.putText(image, text, org, font, fontScale, color, thickness)


image       : It is the image on which text is to be drawn.
text        : Text string to be drawn.
org         : It is the coordinates of the bottom-left corner of the text string in the
              image. The coordinates are represented as tuples of two values i.e. (X
              coordinate value, Y coordinate value).
font        : It denotes the font type. Some of font types are FONT_HERSHEY_SIMPLEX,
              FONT_HERSHEY_PLAIN, , etc.
fontScale   : Font scale factor that is multiplied by the font-specific base size.
color       : It is the color of text string to be drawn. For BGR, we pass a tuple. eg:
              (255, 0, 0) for blue color.
thickness   : It is the thickness of the line in px.
```

shape :-

imread() takes the image as input and decodes into a matrix(array). This array has an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
Synatx:

array_name.shape
```

This returns a tuple containing **(**height, width, number of channels**).**

Number of channels is basically related to RGB.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. **Pyplot** is a collection of functions in the popular visualization package **Matplotlib**. Its functions manipulate elements of a figure, such as creating a figure, creating a plotting area, plotting lines, adding plot labels, etc.

Syntax:

```
# Import matplotlib.pyplot with the alias plt
import matplotlib.pyplot as ____
```

How a computer reads an image?

- Computer sees an image as a matrix of numbers between 0 to 255. For a colored image there will be 3 color channels (R – Red, G- Green, B- Blue).
- There will be a matrix associated with each channel and each element of this matrix represents the intensity of brightness of that pixel.
- All of these channels with have their separate matrices and these will be stacked onto each other to create 3D matrix. So, a computer will interpret a colored image as a 3D matrix.
- **Example:** A colored image of size 700 × 700 means there are 700 rows and 700 columns and 3 channels. **Size of the image will be 700 × 700 × 3**.