INTRODUCTION

1.1 OBJECTIVE

The COVID-19 pandemic has reshaped life as we know it. Many of us are staying home, avoiding people on the street and changing daily habits, like going to school or work, in ways we never imagined.

While we are changing old behaviours, there are new routines we need to adopt. First and foremost is the habit of wearing a mask or face covering whenever we are in a public space to ensure our own safety as well as the safety of others around us.

Perhaps one of the most striking lifestyle changes resulting from the COVID-19 pandemic is the mandatory use of face masks in grocery stores, restaurants and other public places. Wearing a mask, especially when in close proximity to others, is imperative to slowing the spread of COVID-19. But one look outside of your “safe-at-home” haven and you might find people wearing masks in a variety of different styles: dangling from one ear, pulled down below the nose or resting below the chin. These common mistakes decrease the effectiveness of masking and increase the wearer’s risk of catching and spreading the disease.

Many people who have COVID-19 don’t show symptoms but can still spread the virus through droplets that escape from the mouth when speaking, sneezing and coughing. Scientists have proven that masking lowers COVID-19 cases, even in the long term.

Over the past few days, there has been a lot of debate on whether masks should be worn or not worn. Let me first commend the government for their proactive step, albeit late in my opinion, to mandate that masks be worn. If the wearing of masks gives a false sense of security then so too are the actions of washing hands, using hand-sanitizers, wearing gloves, and even the practice of social distancing, when practiced in isolation. The truth is, no single preventative action holds the golden key in disease prevention within the context of proper infection control. Each action contributes significantly to the process and complements the other in disease containment. The lack of protection of face masks is actually a notion touted in part by the World Health Organization (WHO) but with all due respect, the WHO has gotten it wrong more than once. A good example is whether or not the corona virus was airborne. Initially, the WHO stated that the corona virus was not transmittable through the air and later they confirmed that it was. What should guide our action and rather, pro-actions are the issues that uniquely affect our population and common-sense practices. With infectious diseases being responsible for about 25% of the annual global deaths, public health system must engage every possible and viable preventative measure and epidemic control protocols to mitigate against disease transmission.

According to infectious disease expert and senior scholar at the Johns Hopkins Center for Health Security, Amesh A. Adalja, MD, “Face masks can help protect against many respiratory infections that are spread through the droplet route, and that includes coronavirus and the flu.” Speaking to the basis of this recommendation, Dr. Adalja stated that viruses such as the corona virus can spread from an infected person to others through the air by coughing and sneezing or by touching a contaminated surface and then touching your mouth, nose, or eyes prior to hand washing. When a face mask is worn, one can prevent those droplets from coming into contact with one's face or mouth before dropping to the ground.

In an interview with the journal ‘Science’, Chinese Center for Disease Control and Prevention (CDC) Director-General George Gao said on March 27th, “The big mistake in the U.S. and Europe, in my opinion, is that people aren’t wearing masks.”

Several studies from Research Centers in many Ivy League Universities have shown the efficacy of using a DIY (do it yourself) mask made of cotton t-shirts, as a ‘better than no protection’ mode against influenza-like infections.

The transitioning of ‘better than no protection’ to ‘absolute protection’ has been highlighted by an Oxford study, which said that, “Masks are only one component of a complex intervention which must also include eye protection, gowns, behavioural measures to support proper doffing and donning, and general infection control measures.”

In this project, I created a machine learning model that can be used to ensure that people wear masks to prevent the transmission of COVID-19. The model is trained on a dataset of faces with mask and without mask which can be used to classify images of people into categories mentioning with mask and without mask. It can also be used in places where live cameras are installed or real time video streaming to detect faces and classify them as ‘with mask’ or ‘without mask’.

Cameras can be installed in public places, and using this machine learning model we can ensure that people entering in any public place are wearing a mask. It can be installed in entry area of various public places or any place where mask should be worn.

This project is a small step towards public welfare and stopping the spread of deadly corona virus. Following the orders by World Health Organization and government of India is a duty of every citizen but some people violate the rules. To prevent such violations and moreover to prevent the spread of virus, this project will help organizations and responsible people to ensure that every person wears a mask in the places where it is required.

1.2 PROJECT DESCRIPTION

The main goal of the project is to overcome the problem of people not wearing masks in various public places or gatherings. It aims at detecting the faces with mask and without mask using various tools and technologies.
It is implemented as a machine learning model in the form of a code written in python language.
It uses a dataset of various faces with mask and without mask. The dataset is created by taking images of faces and artificially adding a mask to faces by obtaining facial landmarks and applying appropriate algorithms.
The project uses computer vision techniques to read the pixel data from images. This data is converted in the form of features (the pixel data) and labels (a binary value for classification). The machine learning technique requires a set of data for training the model and a small part of it for testing. In this project we have used random forest classifier to classify the images with mask and without mask.

In order to train a custom face mask detector, we need to break our project into two distinct phases, each with its own respective sub-steps:

1. **Training:** Here we'll focus on loading our face mask detection dataset from disk, training a model (using scikit-learn library) on this dataset, and then serializing the face mask detector to disk

2. **Deployment:** Once the face mask detector is trained, we can then move on to loading the mask detector, performing face detection, and then classifying each face as with mask or without mask

After the training of data the model is tested on testing set of data and evaluated on various parameters including accuracy, precision, recall and f1 score.

We deploy the model as such to predict the output from an image of a face with or without mask as well as in real-time video stream using computer vision.

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this.

To capture a video, you need to create a VideoCapture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera.

The video is a collection of frames, each frame treated like images.

The trained model predicts and classifies each frame as 'with mask' and 'without mask'. The output is printed on each frame, all together making it visible as predicted text output in live stream.

2. SYSTEM STUDY

2.1 TOOLS AND TECHNOLOGIES USED

Python - Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming.

Anaconda Navigator - Anaconda is a conditional free and open-source[6] distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator,[13] as a graphical alternative to the command line interface (CLI).

Jupyter Notebook - Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

Numpy – Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Open-CV - OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision.[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez. Advance vision research by providing not only open but also optimized code for basic vision infrastructure.

Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable. Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object detection

Imultis - A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much more easier with OpenCV and both Python 2.7 and Python 3.

Scikit-Learn - Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.[2] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible. Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, numpy for array vectorization, pandas dataframes, scipy, and many more.

- Train test split - Split arrays or matrices into random train and test subsets
- RandomForestClassifier - A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- accuracy_score - In multilabel classification, this function computes subset accuracy
- confusion_matrix - Compute confusion matrix to evaluate the accuracy of a classification.
- precision_recall_fscore_support - Compute precision, recall, F-measure and support for each class

2.2 HARDWARE & SOFTWARE REQUIREMENTS

Webcam – To take video input

- Inbuilt sensitive microphone and image sensor high quality CMOS sensor
- Image resolution interpolated to 25 mega pixels with 6 light sensors
- Image control color saturation, brightness, sharpness and brightness is adjustable; Snap shot switch for taking still pictures
- Anti-flicker 50Hz, 60Hz or outdoor; Resolution hardware: 500K pixels; Image quality: RGB24 or I420
- Exposure: Auto or manual and angle of view: 58 Degree; Interface: USB2.0; Frame rate: 30 fps

Laptop / Desktop – Any Standard laptop / Desktop

3.DATASET

This dataset consists of 1,376 images belonging to two classes:

- with_mask : 690 images
- without_mask : 686 images

The dataset is taken from github repository of Prajna Bhandari.

To create this dataset, Prajna had the ingenious solution of:

1. Taking normal images of faces

2. Then creating a custom computer vision Python script to add face masks to them, thereby creating an artificial (but still real-world applicable) dataset

This method is actually a lot easier than it sounds once you apply facial landmarks to the problem.

Facial landmarks allow us to automatically infer the location of facial structures, including:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

To use facial landmarks to build a dataset of faces wearing face masks, we need to first start with an image of a person not wearing a face mask:

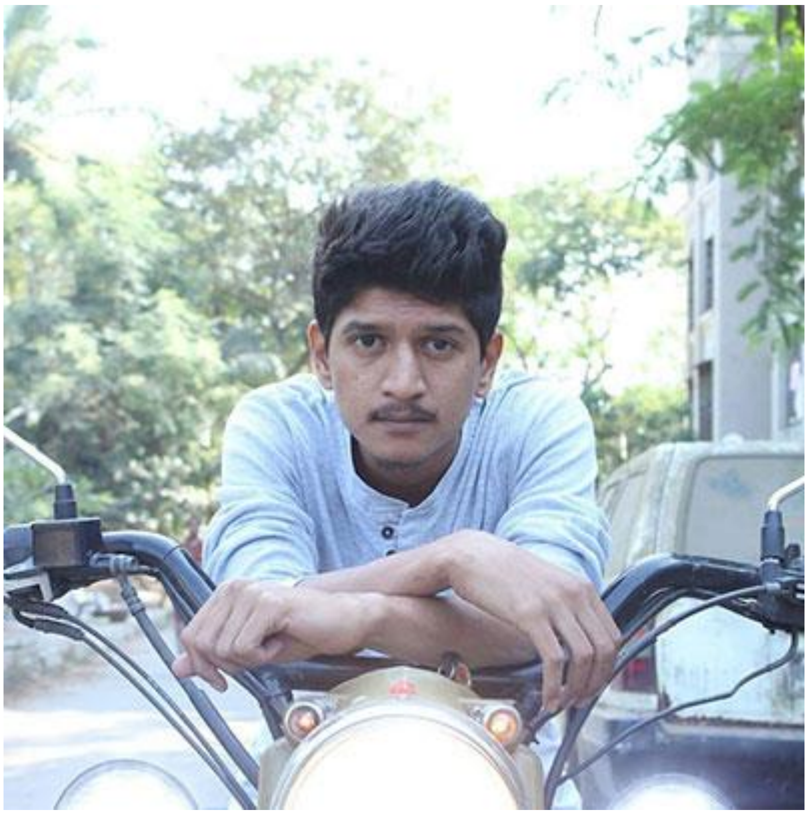


Figure 1: To build a COVID-19/Coronavirus pandemic face mask dataset, we'll first start with a photograph of someone not wearing a face.

From there, we apply face detection to compute the bounding box location of the face in the image:

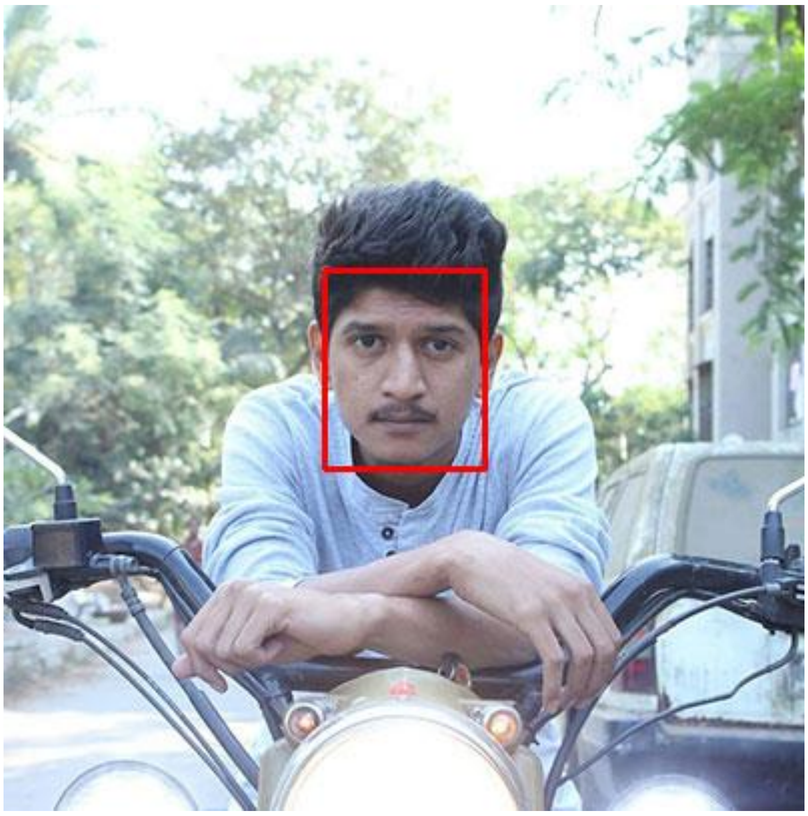


Figure 2: The next step is to apply face detection. Here we've used a deep learning method to perform face detection with OpenCV.

Once we know where in the image the face is, we can extract the face Region of Interest (ROI):

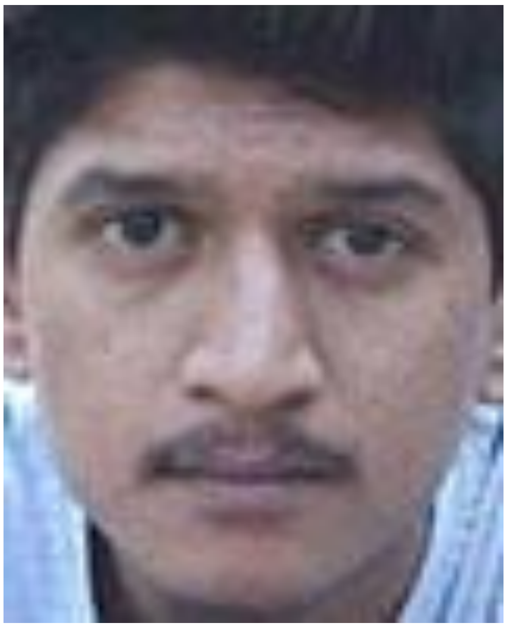


Figure 3: The next step is to extract the face ROI with OpenCV and NumPy slicing.

And from there, we apply facial landmarks, allowing us to localize the eyes, nose, mouth, etc.:

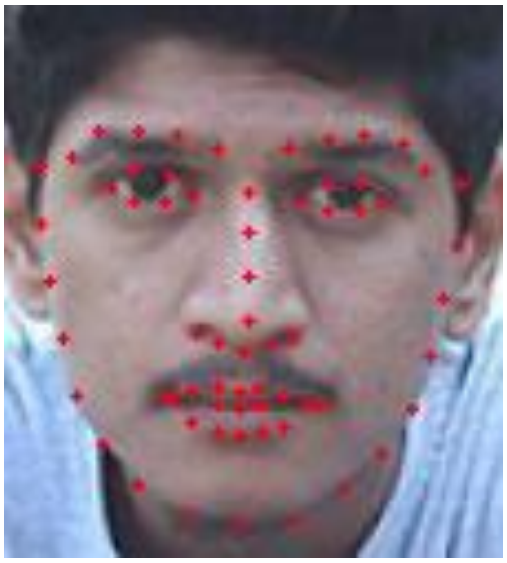


Figure 4: Then, we detect facial landmarks using dlib so that we know where to place a mask on the face.

Next, we need an image of a mask (with a transparent background) such as the one below:

Figure 5: An example of a COVID-19/Coronavirus face mask/shield. This face mask will be overlaid on the original face ROI automatically since we know the face landmark locations.

This mask will be automatically applied to the face by using the facial landmarks (namely the points along the chin and nose) to compute where the mask will be placed.

The mask is then resized and rotated, placing it on the face:

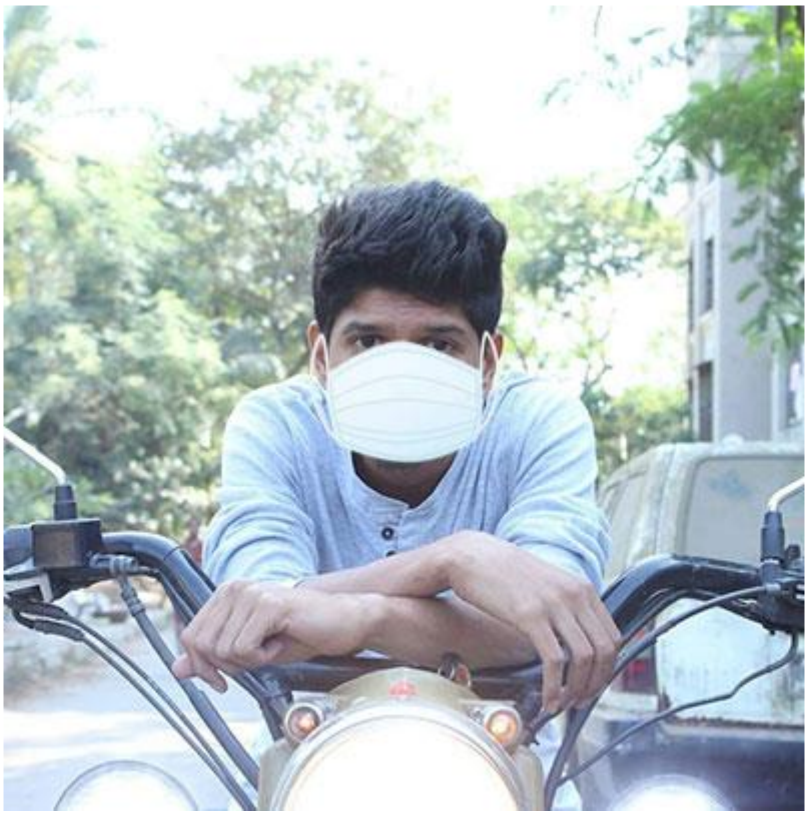


Figure 6: In this figure, the face mask is placed on the person's face in the original frame. It is difficult to tell at a glance that the COVID-19 mask has been applied with computer vision by way of OpenCV and dlib face landmarks.

We can then repeat this process for all of our input images, thereby creating our artificial face mask dataset:

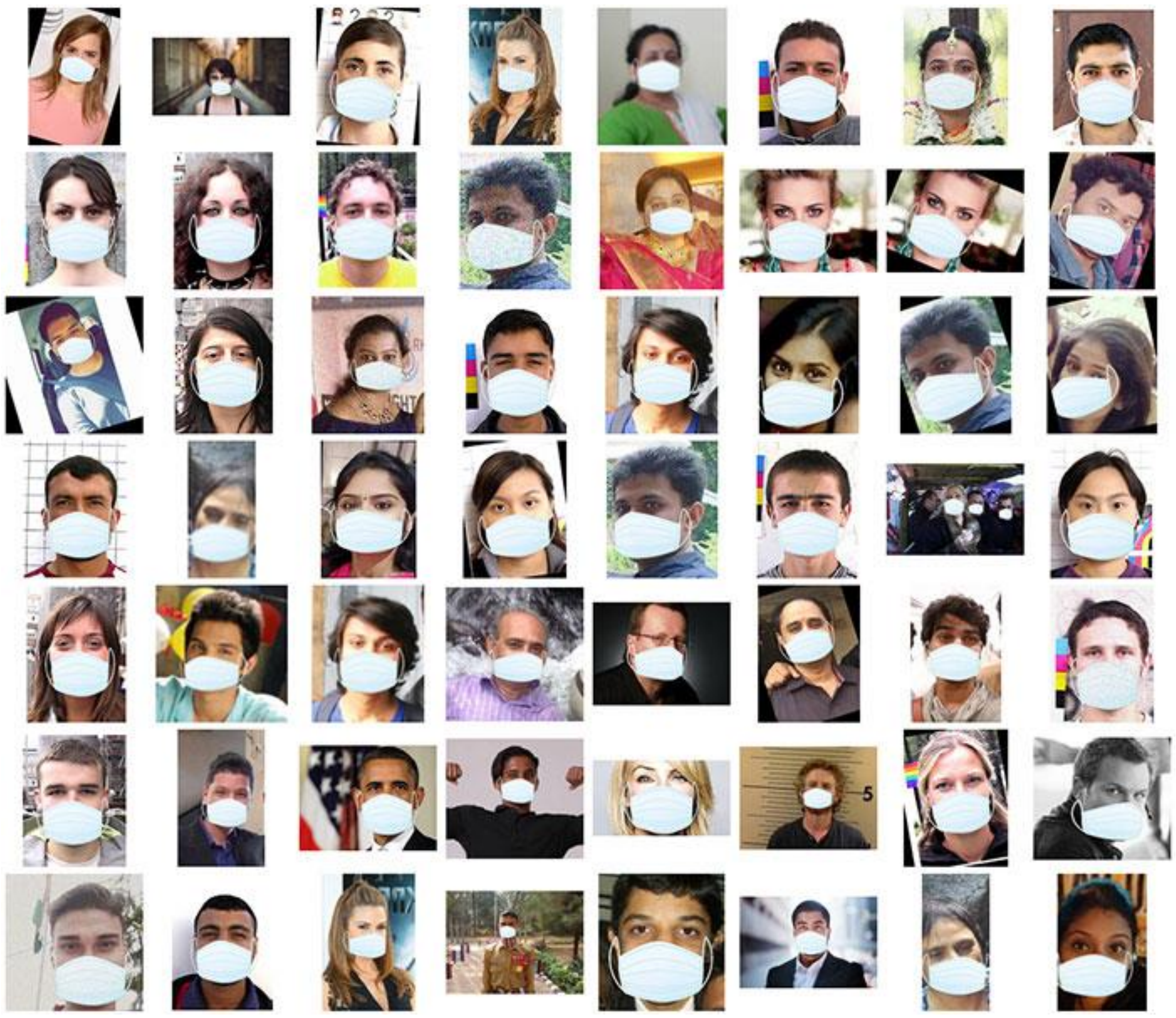
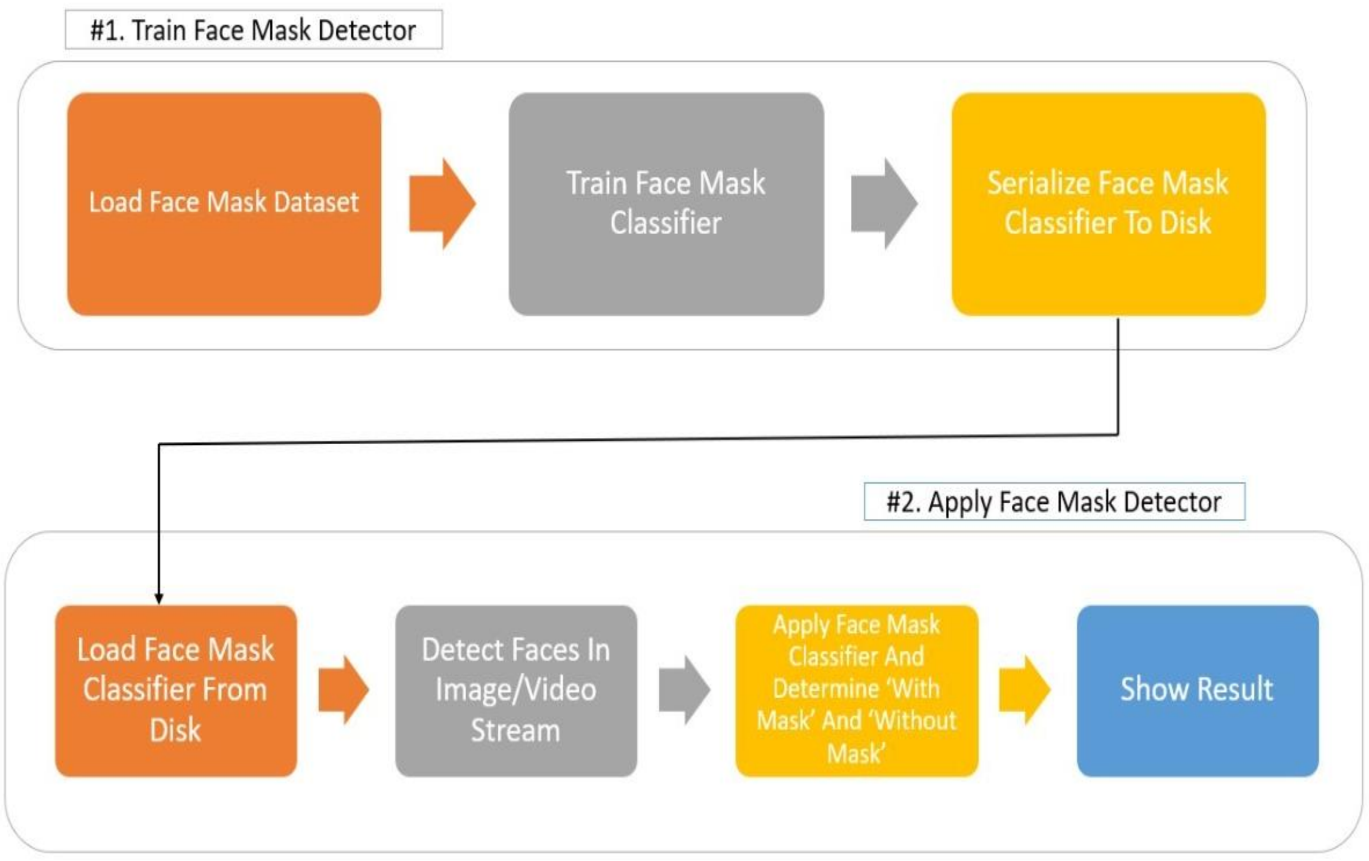


Figure 7: An artificial set of COVID-19 face mask images is shown. This set will be part of our "with mask" / "without mask" dataset for COVID-19 face mask detection with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

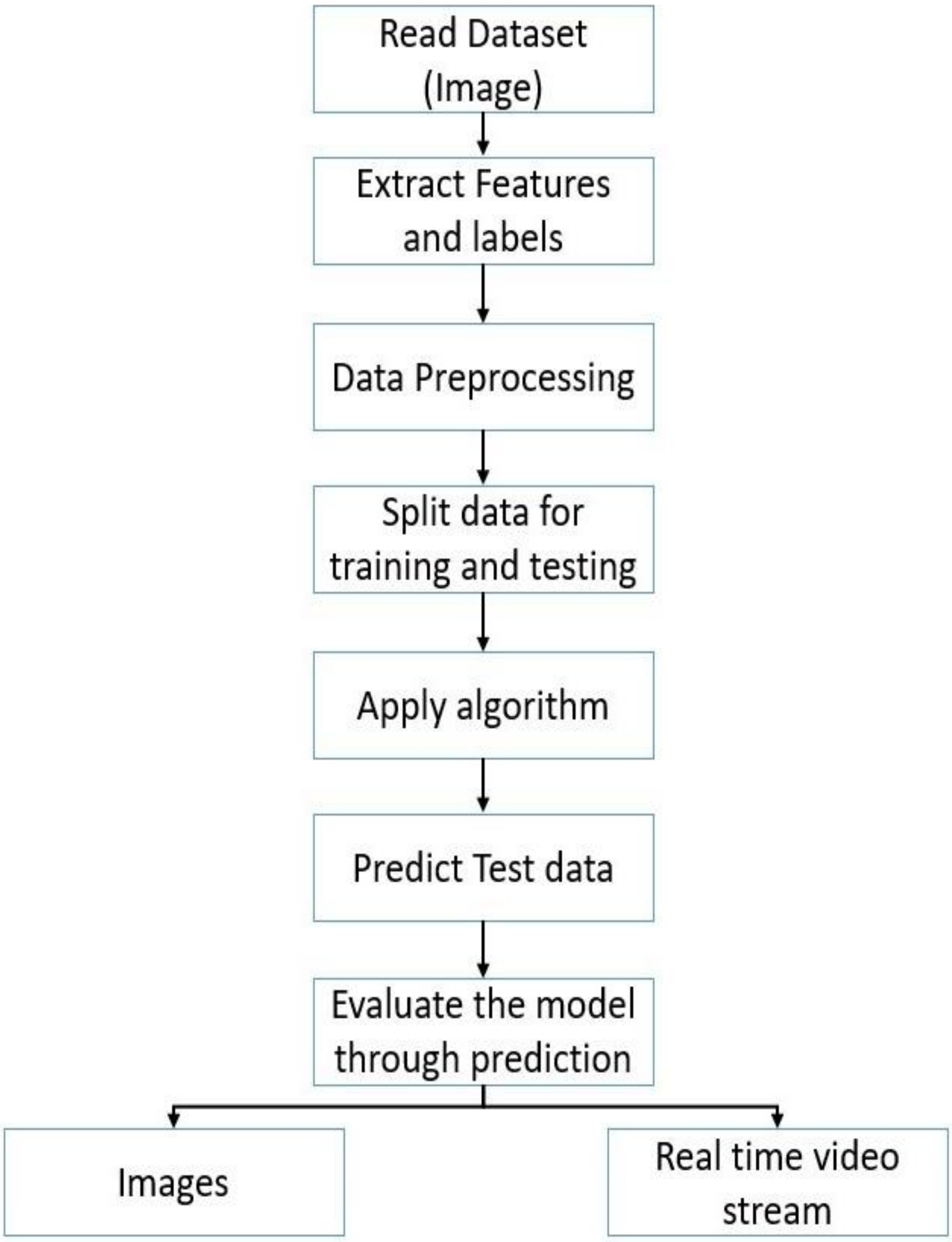
If we include the original images used to generate face mask samples as non-face mask samples, your model will become heavily biased and fail to generalize well. That is avoided at all costs by taking the time to gather new examples of faces without masks.

4. SYSTEM ANALYSIS AND DESIGN

4.1 CONTEXT DIAGRAM (DFD)


#1. Train Face Mask Detector
Load Face Mask Dataset
Train Face Mask Classifier
Serialize Face Mask Classifier To Disk
#2. Apply Face Mask Detector
Load Face Mask Classifier From Disk
Detect Faces In Image/Video Stream
Apply Face Mask Classifier And Determine 'With Mask' And 'Without Mask'
Show Result

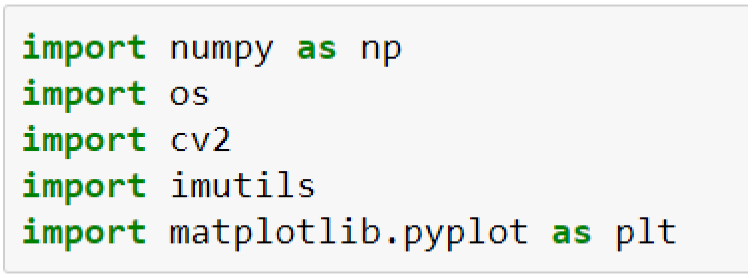
4.2 FLOW CHART


Read Dataset (Image)
Extract Features and labels
Data Preprocessing
Split data for training and testing
Apply algorithm
Predict Test data
Evaluate the model through prediction
Images
Real time video stream

5. IMPLEMENTATION

This project is implemented as a machine learning model coded using python. To accomplish this task, we have used computer vision technique with machine learning algorithm.

An import statement is made up of the import keyword along with the name of the module. In a Python file, this is declared at the top of the code, under any shebang lines or general comments. When we import a module, we are making it available to us in our current program as a separate namespace.



```
import numpy as np
import os
import cv2
import imutils
import matplotlib.pyplot as plt
```

Modules imported in above code consists of numpy, os, cv2, imutils and matplotlib.pyplot.

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. 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.

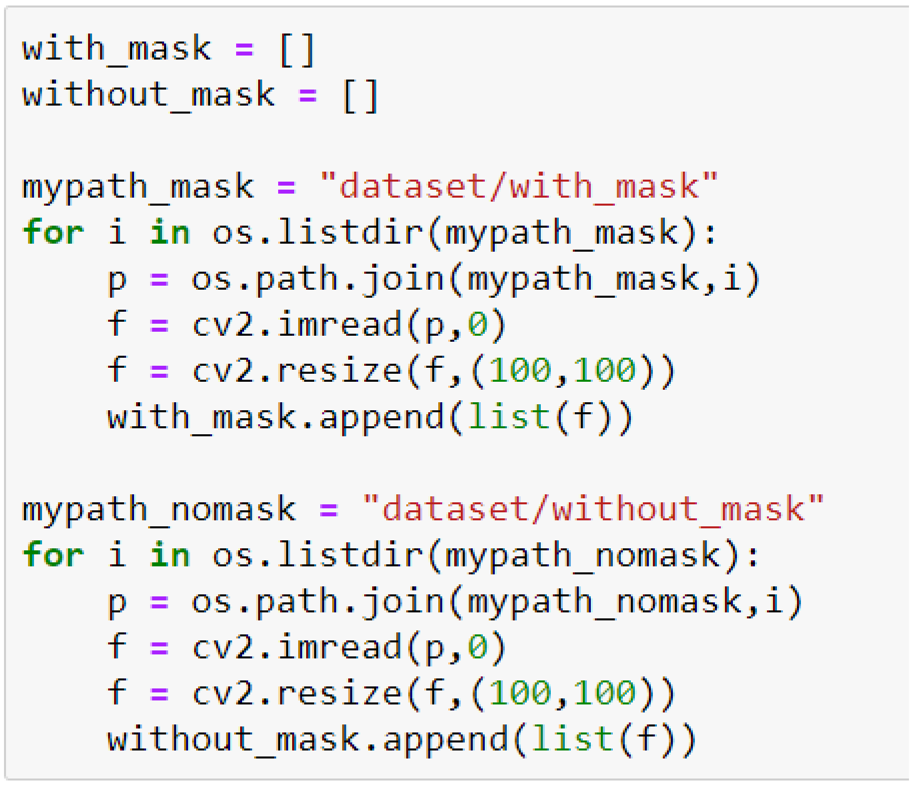
Python OS module provides the facility to establish the interaction between the user and the operating system. It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system. The OS comes under Python's standard utility modules. This module offers a portable way of using operating system dependent functionality. The Python OS module lets us work with the files and directories.

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. To import this we use ‘cv2’ with import statement as mentioned above. It allows us to use different methods available for implementing computer vision techniques.

Imutils are a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3.

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. matplotlib. pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Moving further, we need to extract dataset from a folder to enable its processing in our project.



```
with_mask = []
without_mask = []

mypath_mask = "dataset/with_mask"
for i in os.listdir(mypath_mask):
    p = os.path.join(mypath_mask,i)
    f = cv2.imread(p,0)
    f = cv2.resize(f,(100,100))
    with_mask.append(list(f))

mypath_nomask = "dataset/without_mask"
for i in os.listdir(mypath_nomask):
    p = os.path.join(mypath_nomask,i)
    f = cv2.imread(p,0)
    f = cv2.resize(f,(100,100))
    without_mask.append(list(f))
```

In the above code two arrays named ‘with_mask’ and ‘without_mask’ have been declared.

In the next line there is declaration of a variable of string type, ‘mypath_mask’ consisting of folder address of images with mask. This folder is a sub folder of the dataset folder.

To access the images we use ‘listdir’ method of os module in loop. os. listdir() method in python is used to get the list of all files and directories in the specified directory. If we don't specify any directory, then list of files and directories in the current working directory will be returned.

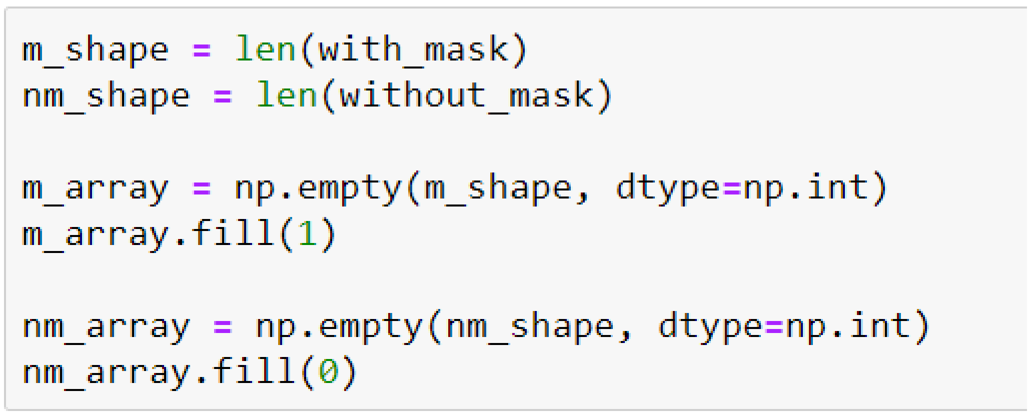
Further, *os.path.join()* method in Python join one or more path components intelligently. This method concatenates various path components with exactly one directory separator (‘/’) following each non-empty part except the last path component. If the last path component to be joined is empty then a directory separator (‘/’) is put at the end.
If a path component represents an absolute path, then all previous components joined are discarded and joining continues from the absolute path component.

Once the list of image files is created it is needed to extract data from images that can be used with appropriate algorithms. 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.

The data read from the image is stored in a variable each time, while in a loop, but the images may be of different size and dimensions which is needed to be same. For resizing of image, we use cv2.resize() method. Image resizing refers to the scaling of images. Scaling comes handy in many image processing as well as machine learning applications. It helps in reducing the number of pixels from an image and that has several advantages e.g. It can reduce the time of training of a neural network as more is the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model.

After the image data has been extracted from the images the data is then added to the declared list 'with_mask'.

Similarly, the 'without_mask' list is also created consisting of the unmasked images' data.



```
m_shape = len(with_mask)
nm_shape = len(without_mask)

m_array = np.empty(m_shape, dtype=np.int)
m_array.fill(1)

nm_array = np.empty(nm_shape, dtype=np.int)
nm_array.fill(0)
```

In the above lines we are using numpy library to create 2 arrays having values 0 and 1 as all elements in each array.

len() is a built-in function in python. It can be used to get the length of the given string, array, list, tuple, dictionary, etc. ... Return value a return an integer value i.e. the length of the given string, or array, or list, or collections.

The empty() function is used to create a new array of given shape and type, without initializing entries.

fill() method is used to fill the numpy array with a scalar value. If we have to initialize a numpy array with an identical value then we use numpy. ndarray. fill().

The purpose of creating this list is to assign some lable to the images with mask and without mask.

In machine learning, if you have labeled data, that means your data is marked up, or annotated, to show the target, which is the answer you want your machine learning model to predict. In general, data labelling can refer to tasks that include data tagging, annotation, classification, moderation, transcription, or processing.

```
features = with_mask + without_mask
label = np.concatenate((m_array,nm_array))
plt.imshow(features[0])
```

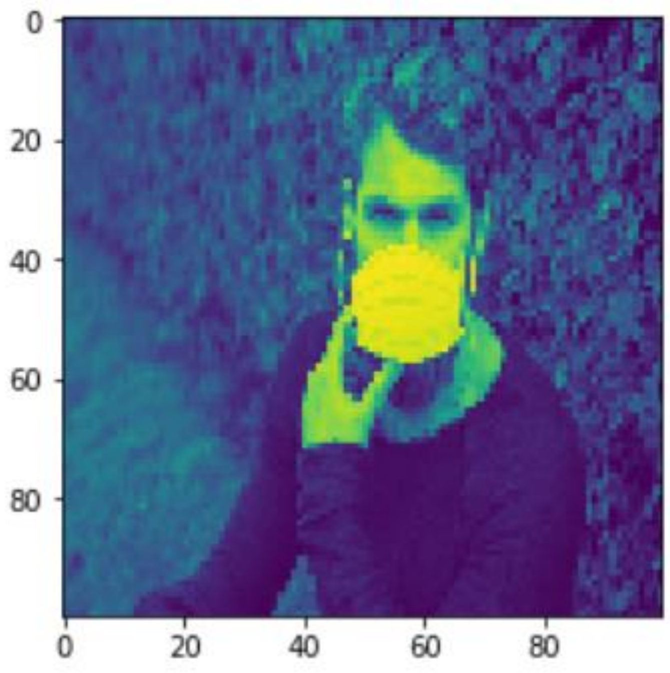
Features are individual independent variables that act as the input in your system. Prediction models use features to make predictions. New features can also be obtained from old features using a method known as ‘feature engineering’. More simply, you can consider one column of your data set to be one feature. Sometimes these are also called attributes. And the number of features are called dimensions.

Labels are the final output. You can also consider the output classes to be the labels. When data scientists speak of labelled data, they mean groups of samples that have been tagged to one or more labels.

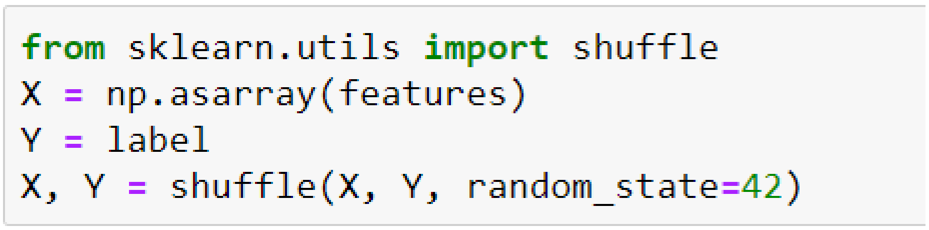
To create labels we concatenated i.e., combines the two arrays containing numbers 0 and 1. The feature array is creates by concatenating the two lists containing images with mask and without mask. The labels are created in such a way that in the array there are equal number of 1’s as the number of images with mask and equal number of 0’s as the number of images without mask in a sequence respectively.

The matplotlib function `imshow()` creates an image from a 2-dimensional numpy array. The image will have one square for each element of the array. The color of each square is determined by the value of the corresponding array element and the color map used by `imshow()`. The above lines will give the following output:

```
<matplotlib.image.AxesImage at 0x238786bccf8>
```



The image displayed is the 1st image from the dataset.



```
from sklearn.utils import shuffle
X = np.asarray(features)
Y = label
X, Y = shuffle(X, Y, random_state=42)
```

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

We put the features in a variable named X and labels in a variable named Y.

In machine learning (ML), we are often presented with a dataset that will be further split into training, testing & validation datasets. It is very important that dataset is shuffled well to avoid any element of bias/patterns in the split datasets before training the ML model.

Shuffling the data improves the ML model quality and it also improves the predictive performance.

We use sklearn.utils.shuffle(*arrays, **options) to perform shuffle arrays or sparse matrices in a consistent way.

This is a convenience alias to resample(*arrays, replace=False) to do random permutations of the collections.

Parameters: *arrays : sequence of indexable data-structures

Indexable data-structures can be arrays, lists, dataframes or scipy sparse matrices with consistent first dimension.

Returns: shuffled_arrays : sequence of indexable data-structures

Sequence of shuffled copies of the collections. The original arrays are not impacted.

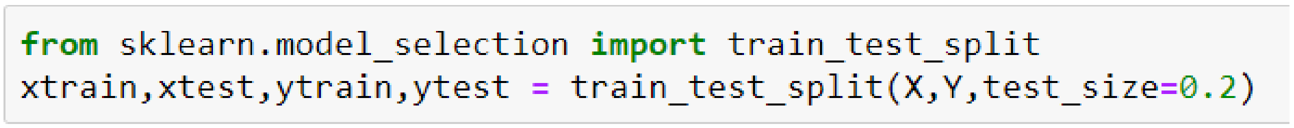
Other Parameters:

random_state : int, RandomState instance or None, optional (default=None)

The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

n_samples : int, None by default

Number of samples to generate. If left to None this is automatically set to the first dimension of the arrays.



```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(X,Y,test_size=0.2)
```

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

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

It can be used for classification or regression problems and can be used for any supervised 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.

- 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.

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. Here, we have used test_size=0.2 which means the data is split into 80% training data and 20% testing data.

The method used is:

sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)

It split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

Parameters

*arrayssequence of indexables with same length / shape[0]

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_sizefloat or int, default=None

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If train_size is also None, it will be set to 0.25.

train_sizefloat or int, default=None

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_stateint, RandomState instance or None, default=None

Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shufflebool, default=True

Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratifyarray-like, default=None

If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

Returns

splittinglist, length=2 * len(arrays)

List containing train-test split of inputs.

New in version 0.16: If the input is sparse, the output will be a scipy.sparse.csr_matrix. Else, output type is the same as the input type.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

After the processing of dataset and extracting features and labels, we searched for the appropriate algorithm to train the data. Firstly, the project is based in image classification so we decided to use any standard classifier algorithm that gives best results. We started with SVM and KNN classifiers.

1. SVM (Support Vector Machine) –
    - Not good in accuracy.
    - Time taken (up to 2 to 3 hrs. for training).
2. KNN (K-nearest neighbor algorithm) –
    - Less time but not good in accuracy.

Next after analysing the project requirement and researching we moved to the ensemble methods (for random forest classifier). It gave more accuracy in lesser amount of time which is in turn the best standard algorithm we could use according to the scenario.

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

The sklearn.ensemble module includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-Trees method. Both algorithms are perturb-and-combine techniques [B1998] specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.

In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features. (See the parameter tuning guidelines for more details).

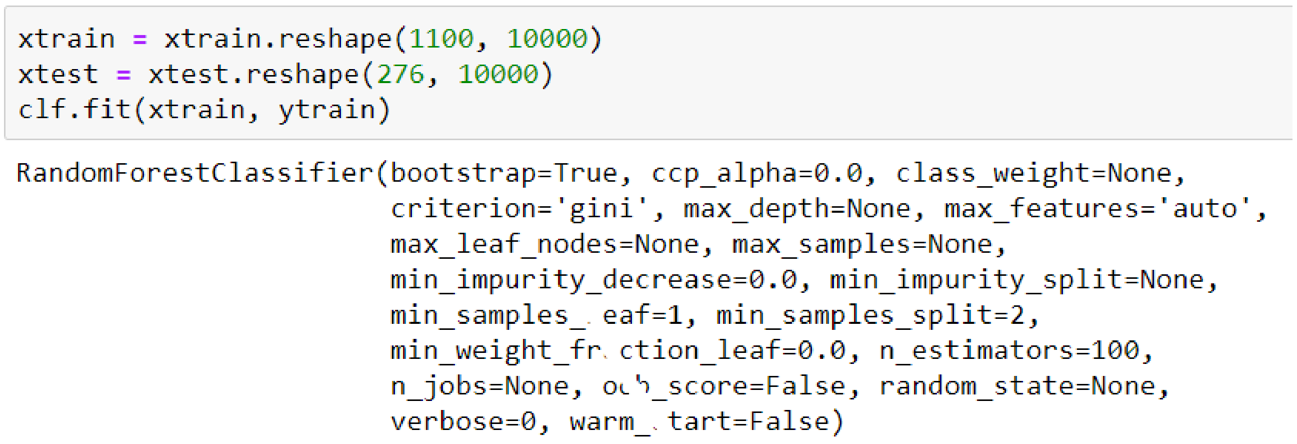
The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model.

In contrast to the original publication [B2001], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

The main parameters to adjust when using these methods is n_estimators and max_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max_features=None (always considering all features instead of a random subset) for regression problems, and max_features="sqrt" (using a random subset of size sqrt(n_features)) for classification tasks (where n_features is the number of features in the data). Good results are often achieved when setting max_depth=None in combination with min_samples_split=2 (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (bootstrap=True) while the default strategy for extra-trees is to use the whole dataset (bootstrap=False). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting oob_score=True.

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=1e-07,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

Once the algorithm is selected the model needs to be trained using the classifier.



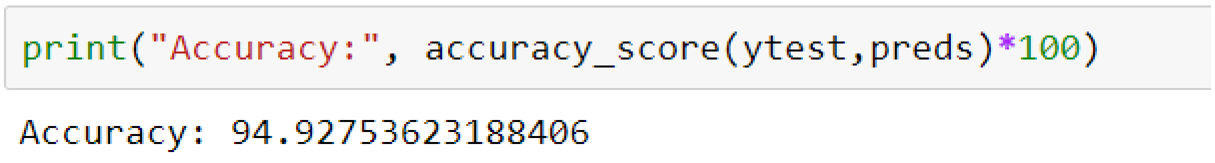
```
xtrain = xtrain.reshape(1100, 10000)
xtest = xtest.reshape(276, 10000)
clf.fit(xtrain, ytrain)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In terms of machine learning, Clf is an estimator instance, which is used to store model.

We use clf to store trained model values, which are further used to predict value, based on the previously stored weights.

The fit() method takes the training data as arguments, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning. Note that the model is fitted using X and y , but the object holds no reference to X and y .



```
print("Accuracy:", accuracy_score(ytest,preds)*100)
```

```
Accuracy: 94.92753623188406
```

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right. Formally, accuracy has the following definition:

Accuracy=Number of correct predictions / Total number of predictions

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Accuracy=(TP+TN) / (TP+TN+FP+FN)

Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

We use sklearn.metrics.**accuracy_score**(*y_true*, *y_pred*, *, *normalize=True*, *sample_weight=None*) to calculate accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true.

Parameters

y_true 1d array-like, or label indicator array / sparse matrix

Ground truth (correct) labels.

y_pred 1d array-like, or label indicator array / sparse matrix

Predicted labels, as returned by a classifier.

Normalize bool, default=True

If False, return the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples.

sample_weight array-like of shape (n_samples,), default=None

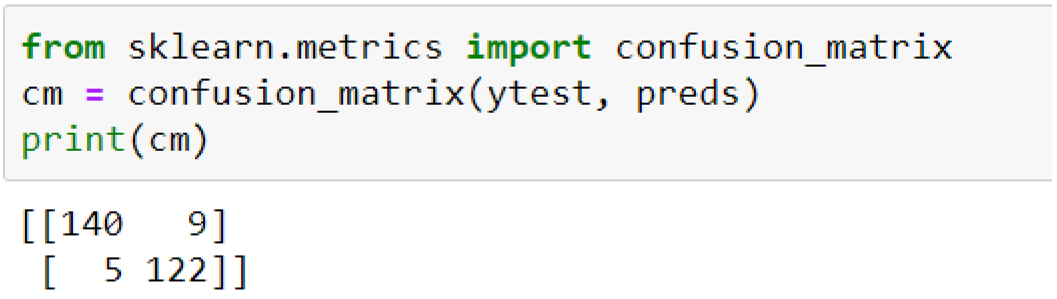
Sample weights.

Returns

Score float

If normalize == True, return the fraction of correctly classified samples (float), else returns the number of correctly classified samples (int).

The best performance is 1 with normalize == True and the number of samples with normalize == False.



```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, preds)
print(cm)
```

```
[[140   9]
 [  5 122]]
```

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).[8] The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabelling one as another).

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

In abstract terms, the confusion matrix is as follows:

Predicted	Actual class	
Class	PN	
P	TP	FP
N	FN	TN

where: P = Positive;

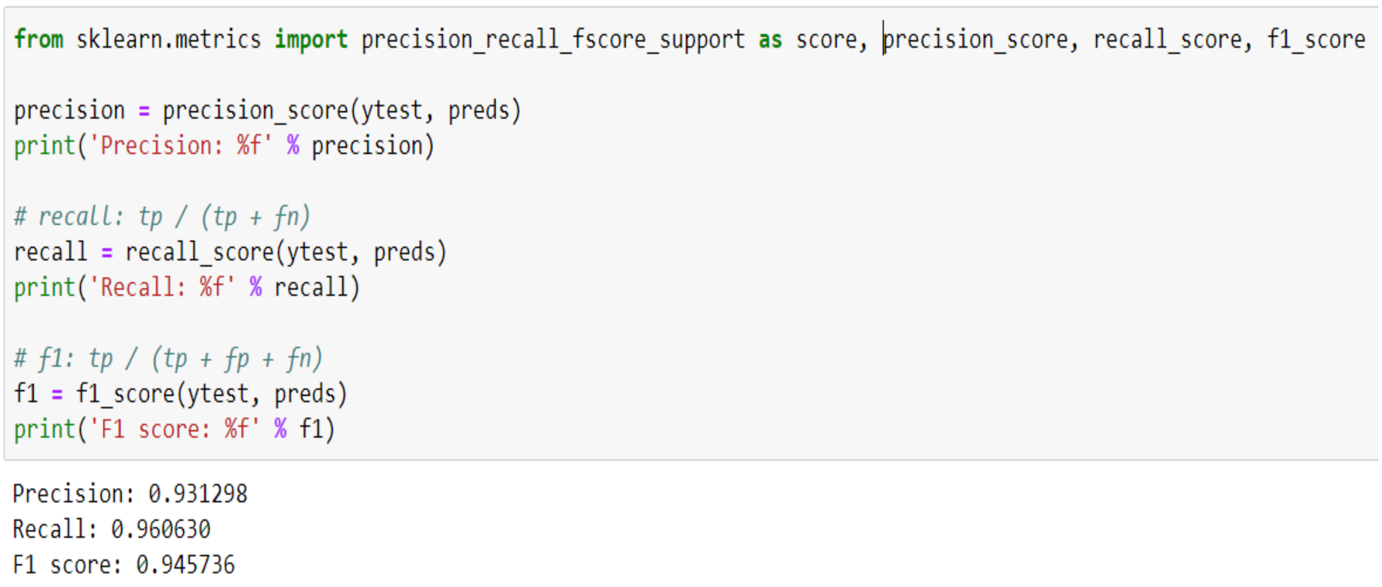
N = Negative;

TP = True Positive;

FP = False Positive;

TN = True Negative;

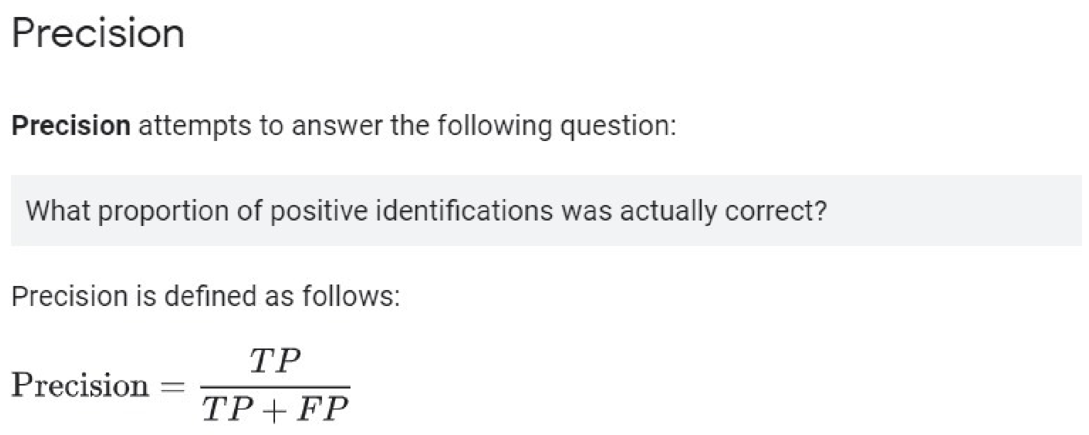
FN = False Negative.



```
from sklearn.metrics import precision_recall_fscore_support as score, precision_score, recall_score, f1_score

precision = precision_score(ytest, preds)
print('Precision: %f' % precision)

# recall: tp / (tp + fn)
recall = recall_score(ytest, preds)
print('Recall: %f' % recall)

# f1: tp / (tp + fp + fn)
f1 = f1_score(ytest, preds)
print('F1 score: %f' % f1)
```

```
Precision: 0.931298
Recall: 0.960630
F1 score: 0.945736
```

In pattern recognition, information retrieval and classification (machine learning), precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of the total amount of relevant instances that were actually retrieved. Both precision and recall are therefore based on an understanding and measure of relevance.



Precision

Precision attempts to answer the following question:

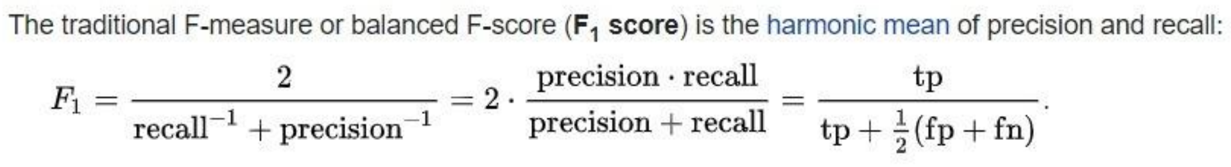
What proportion of positive identifications was actually correct?

Precision is defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

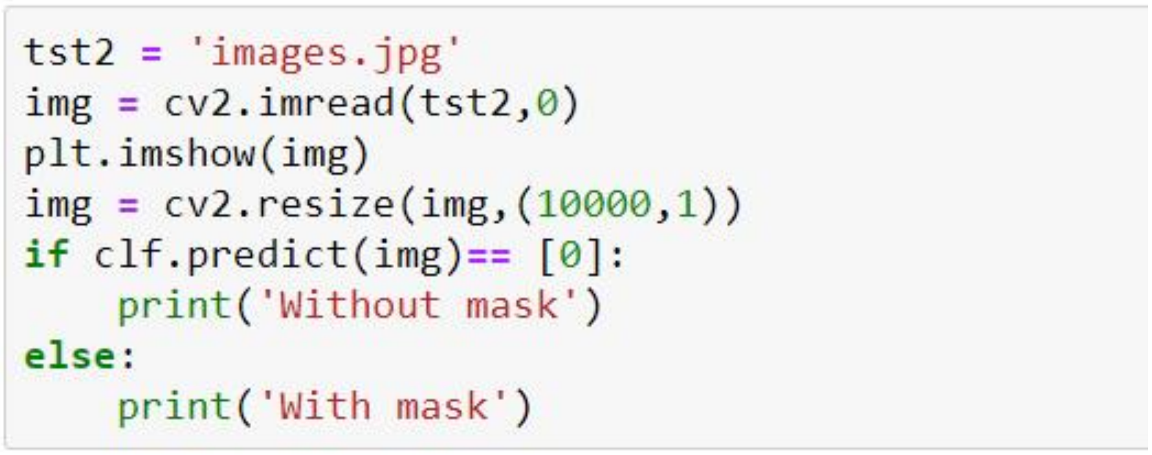
The F1 score is the harmonic mean of the precision and recall. The more generic F_score applies additional weights, valuing one of precision or recall more than the other.

The highest possible value of an F-score is 1, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero. The F1 score is also known as the Sørensen–Dice coefficient or Dice similarity coefficient (DSC).



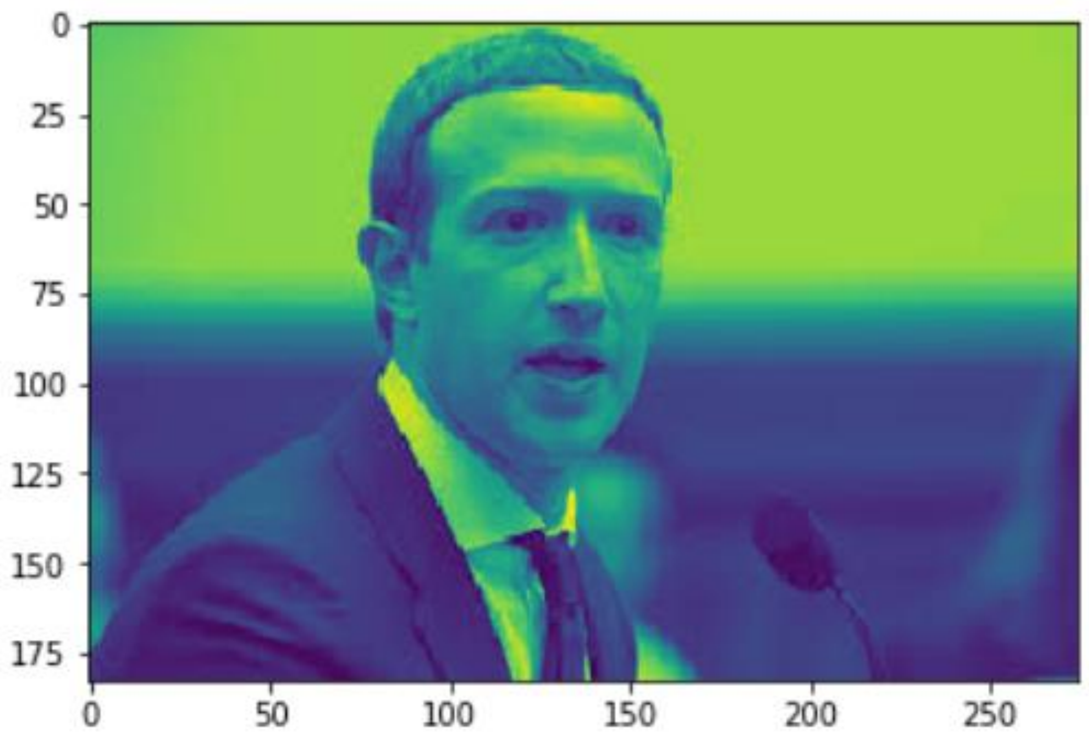
The traditional F-measure or balanced F-score (**F_1 score**) is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$



```
tst2 = 'images.jpg'
img = cv2.imread(tst2,0)
plt.imshow(img)
img = cv2.resize(img,(10000,1))
if clf.predict(img)== [0]:
    print('Without mask')
else:
    print('With mask')
```

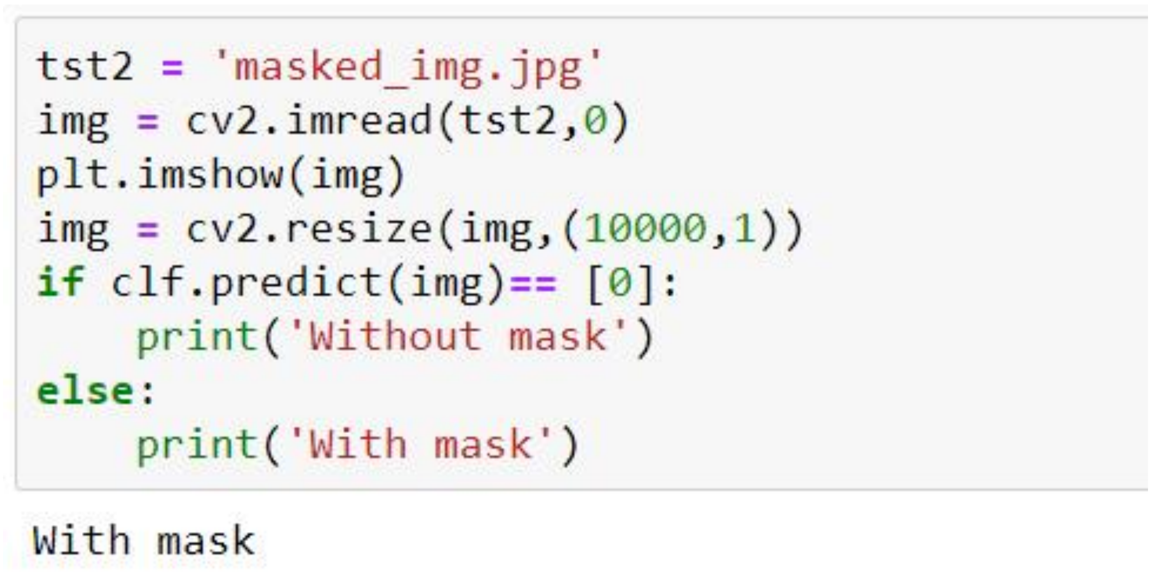
```
Without mask
```



In the above lines of code, a random image is picked apart from the dataset and tested for classification. The classifier predicts the image as with mask or without mask.

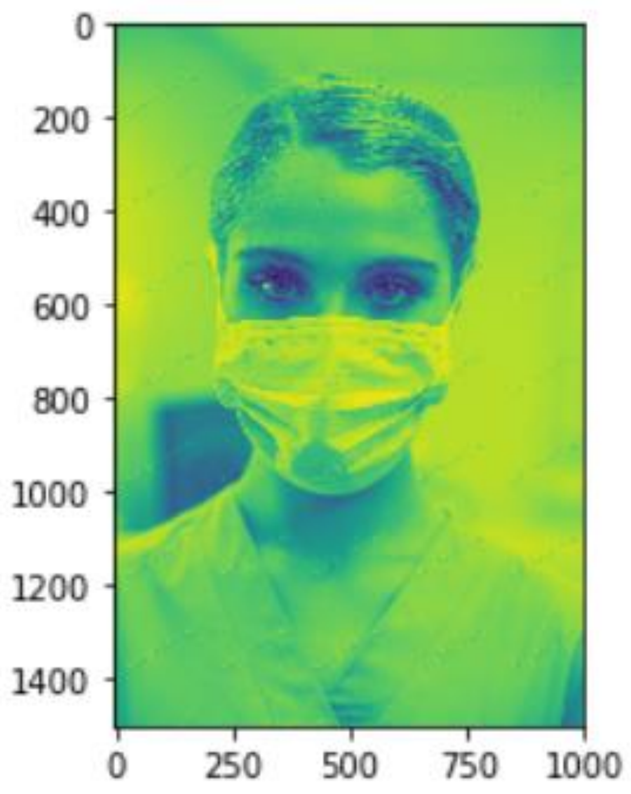
To implement this image prediction, firstly the selected image is passed through the cv2.imread() method which extracts the pixel data from the image in the form of array just like that in the features part of our dataset used.

If the classifier gives the predicted value as 0 it means the image is without mask whereas if the classifier predicts the value 1, it means that the image is with mask.
Here, as seen the face of the person is without mask and so is the prediction of the classifier.

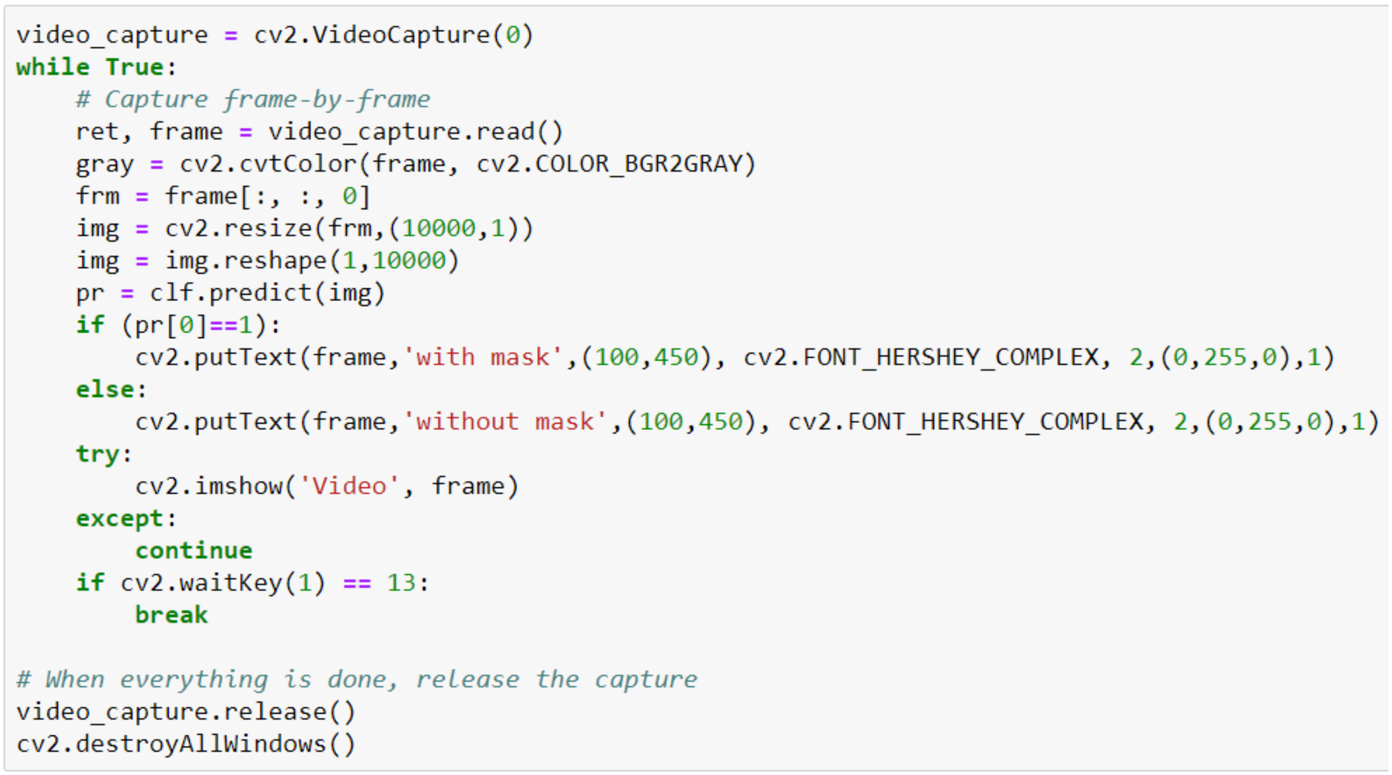


```
tst2 = 'masked_img.jpg'
img = cv2.imread(tst2,0)
plt.imshow(img)
img = cv2.resize(img,(10000,1))
if clf.predict(img)== [0]:
    print('Without mask')
else:
    print('With mask')
```

```
With mask
```



The above image prediction is another example but this time we have used the image in which the person is wearing the mask. The classifier has predicted value 1 and so the result displayed is ‘with mask’.
Also in both the cases the image has been plotted using the imshow() from matplotlib.pyplot.



```python
video_capture = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frm = frame[:, :, 0]
    img = cv2.resize(frm,(10000,1))
    img = img.reshape(1,10000)
    pr = clf.predict(img)
    if (pr[0]==1):
        cv2.putText(frame,'with mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    else:
        cv2.putText(frame,'without mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    try:
        cv2.imshow('Video', frame)
    except:
        continue
    if cv2.waitKey(1) == 13:
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

We deploy the model as such to predict the image output as well as in real-time video using computer vision.

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this. To capture a video, you need to create a VideoCapture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera.

You can use VideoCapture() method of cv2 library to read and start live streaming. To use cv2 library, you need to import cv2 library using import statement.
It is used as cv2.VideoCapture(video_path or device index)
Parameters
There is one argument which needs to pass in VideoCapture() class.

video_path: Locationvideo in your system in string form with their extensions like .mp4, .avi, etc.
OR
device index: It is just the number to specify the camera. Its possible values ie either 0 or -1.

The video is a collection of frames, each frame treated like images.
The trained model predicts and classifies each frame as 'with mask' and 'without mask'.
The output is printed on each frame, all together making it visible as predicted text output in live stream.

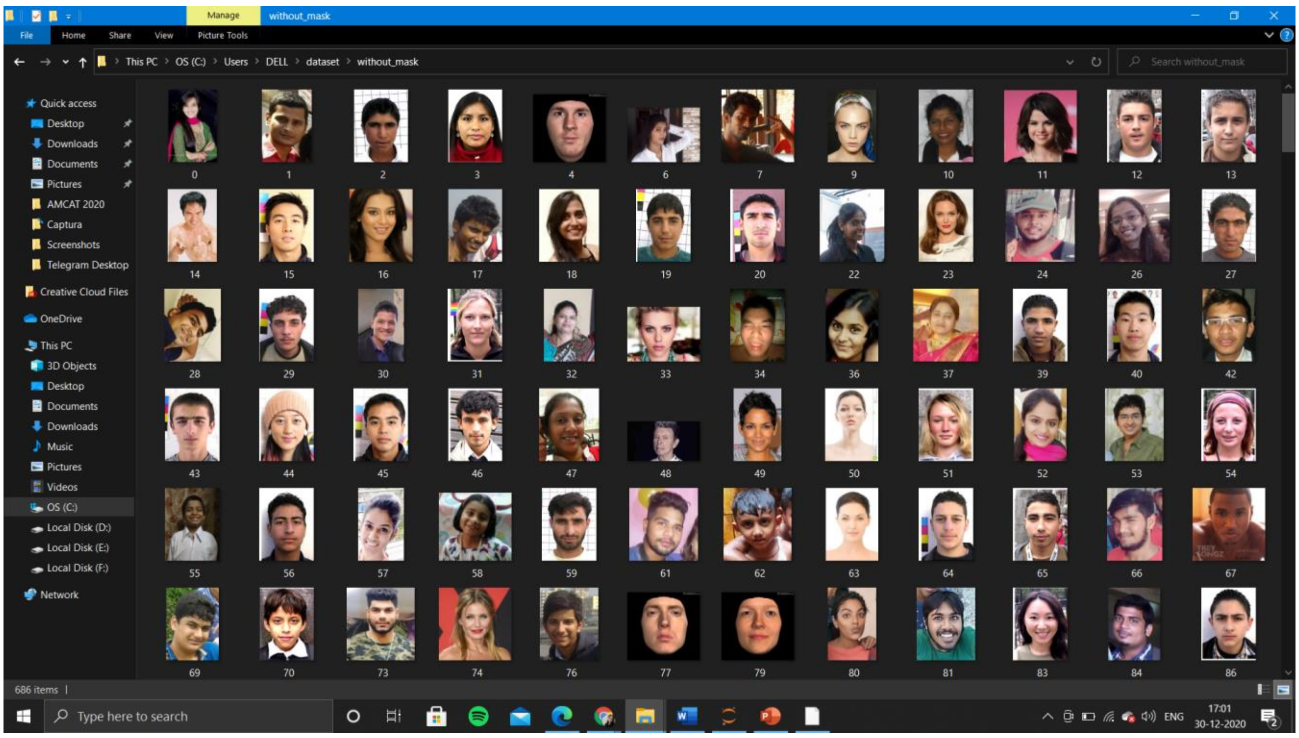
while(True): **ret**, **frame** = **cap**. **read()** This code initiates an infinite loop (to be broken later by a break statement), where we have **ret** and **frame** being defined as the **cap**. **read()**. Basically, **ret** is a Boolean regarding whether or not there was a return at all, at the frame is each frame that is returned.

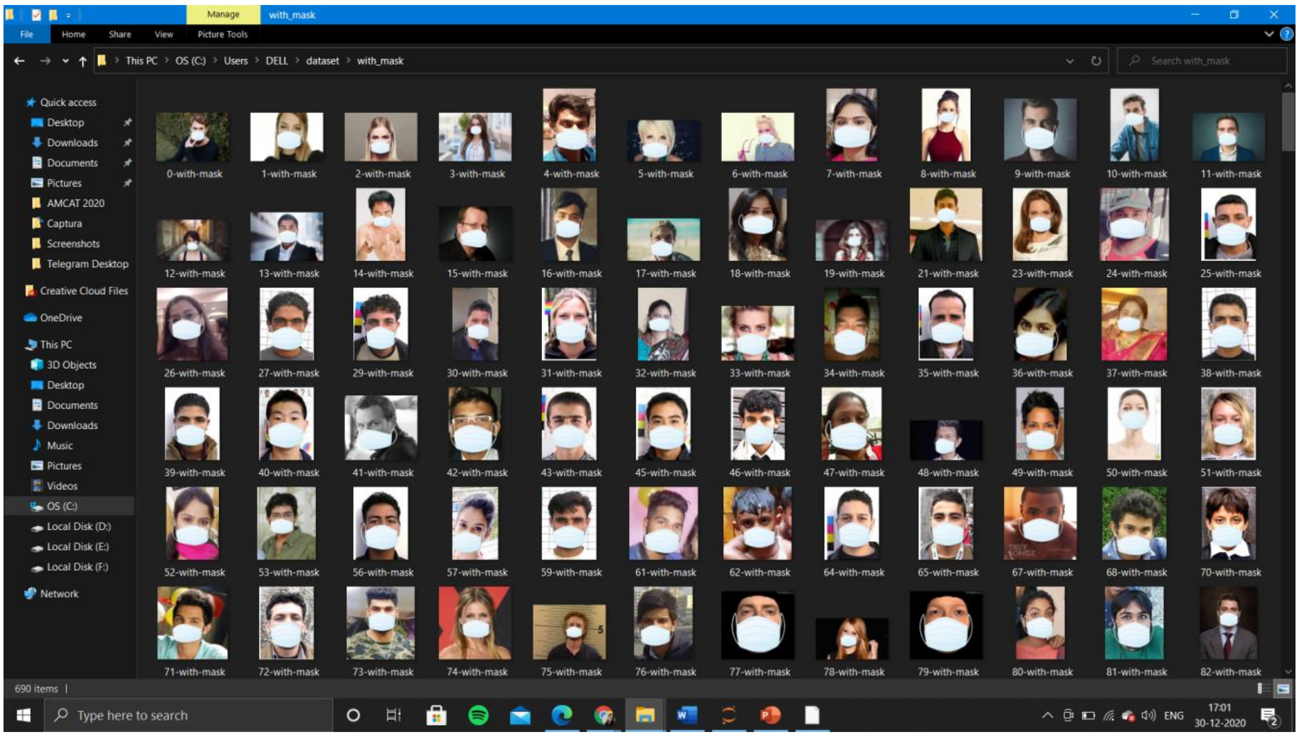
Then, we will need to obtain the frames of our camera one by one and convert them to gray. This means we should write that logic inside an infinite loop, so we keep obtaining frames, converting them to gray and displaying them. Pass the frame to the cv2.cvtColor**()** method with cv2.COLOR_BGR2GRAY as a parameter to convert it into gray-scale.

The classifier predicts the output for each frame and classify it accordingly. To show the predicted result in output stream a text is added to each frame stating 'with mask' or 'without mask' based on the prediction. It is done using cv2.putText() method. This method is used to draw a text string on any image.

SCREENSHOTS

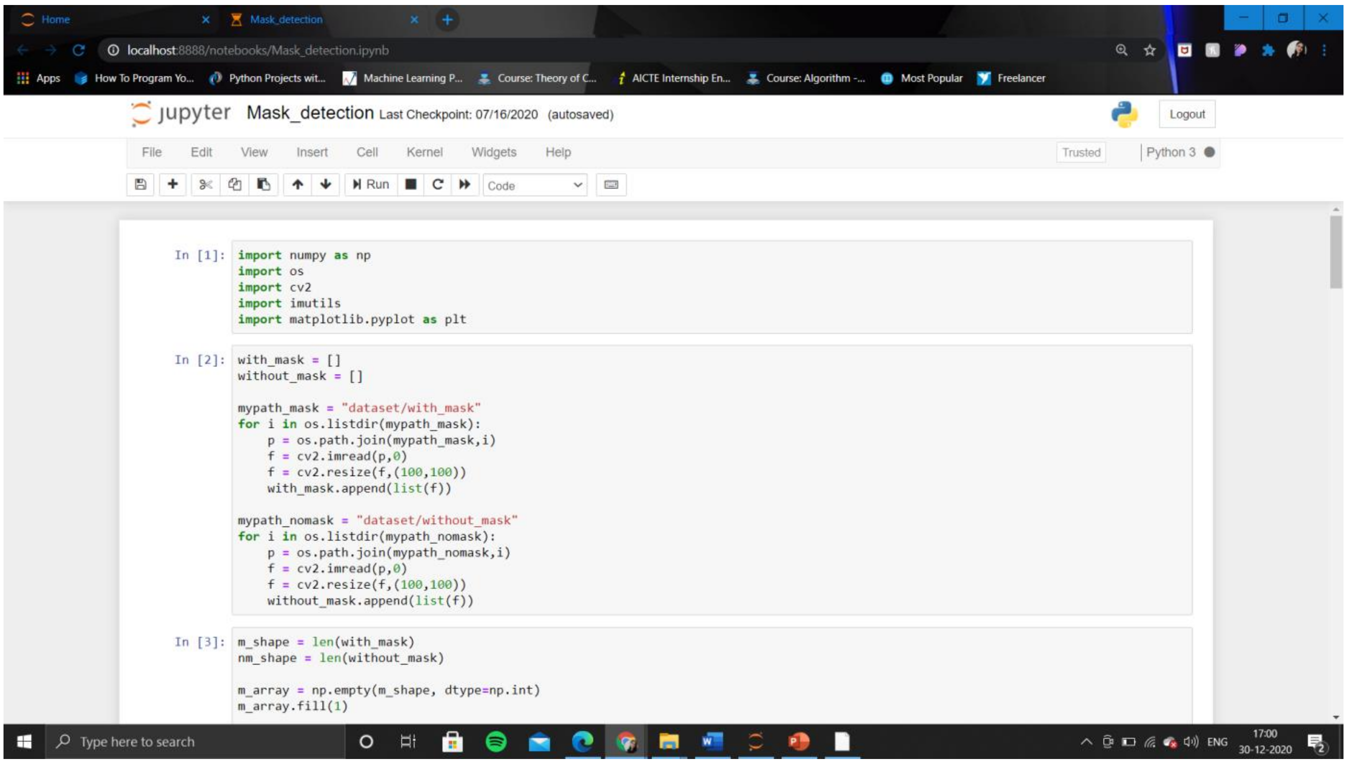
Dataset





with_mask
This PC > OS (C:) > Users > DELL > dataset > with_mask
0-with-mask 1-with-mask 2-with-mask 3-with-mask 4-with-mask 5-with-mask 6-with-mask 7-with-mask 8-with-mask 9-with-mask 10-with-mask 11-with-mask
12-with-mask 13-with-mask 14-with-mask 15-with-mask 16-with-mask 17-with-mask 18-with-mask 19-with-mask 21-with-mask 23-with-mask 24-with-mask 25-with-mask
26-with-mask 27-with-mask 29-with-mask 30-with-mask 31-with-mask 32-with-mask 33-with-mask 34-with-mask 35-with-mask 36-with-mask 37-with-mask 38-with-mask
39-with-mask 40-with-mask 41-with-mask 42-with-mask 43-with-mask 45-with-mask 46-with-mask 47-with-mask 48-with-mask 49-with-mask 50-with-mask 51-with-mask
52-with-mask 53-with-mask 56-with-mask 57-with-mask 59-with-mask 61-with-mask 62-with-mask 64-with-mask 65-with-mask 67-with-mask 68-with-mask 70-with-mask
71-with-mask 72-with-mask 73-with-mask 74-with-mask 75-with-mask 76-with-mask 77-with-mask 78-with-mask 79-with-mask 80-with-mask 81-with-mask 82-with-mask
690 items

Code

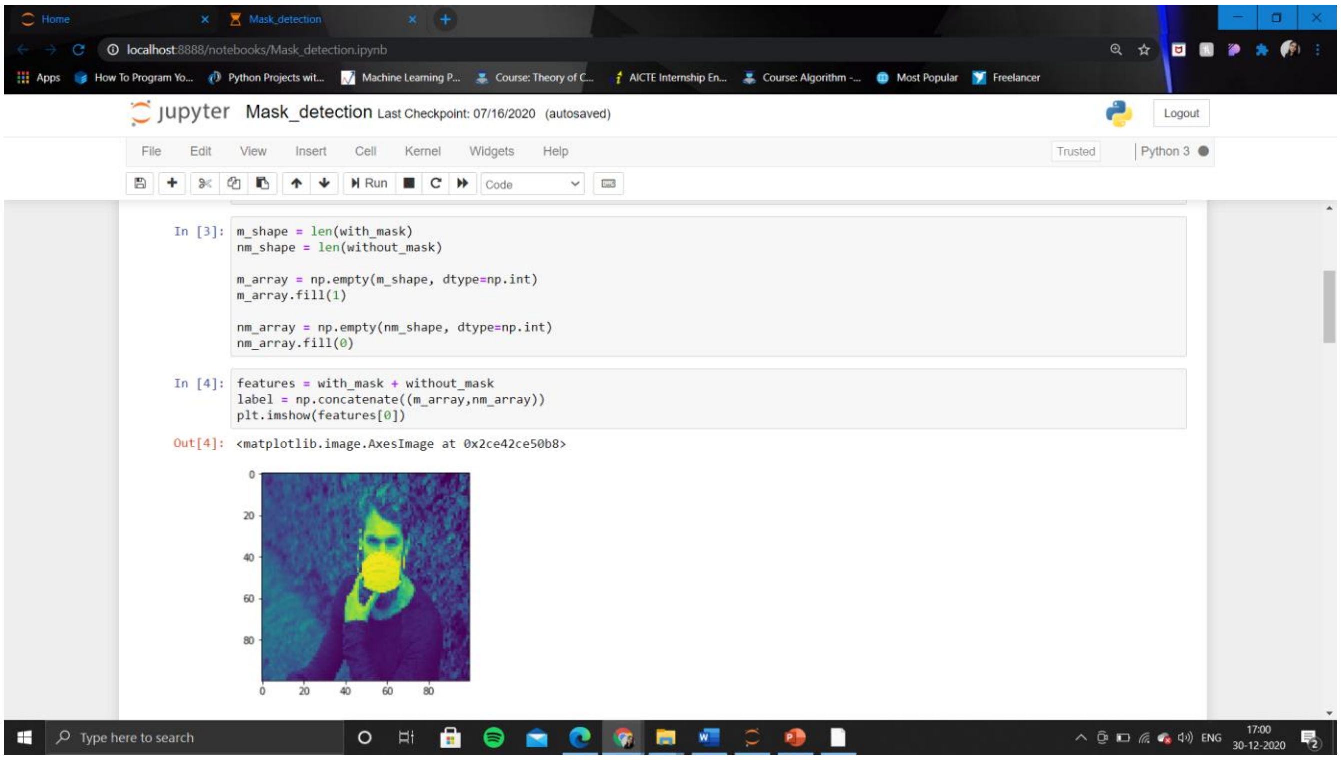


localhost:8888/notebooks/Mask_detection.ipynb
jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3

```
In [1]: import numpy as np
        import os
        import cv2
        import imutils
        import matplotlib.pyplot as plt
```

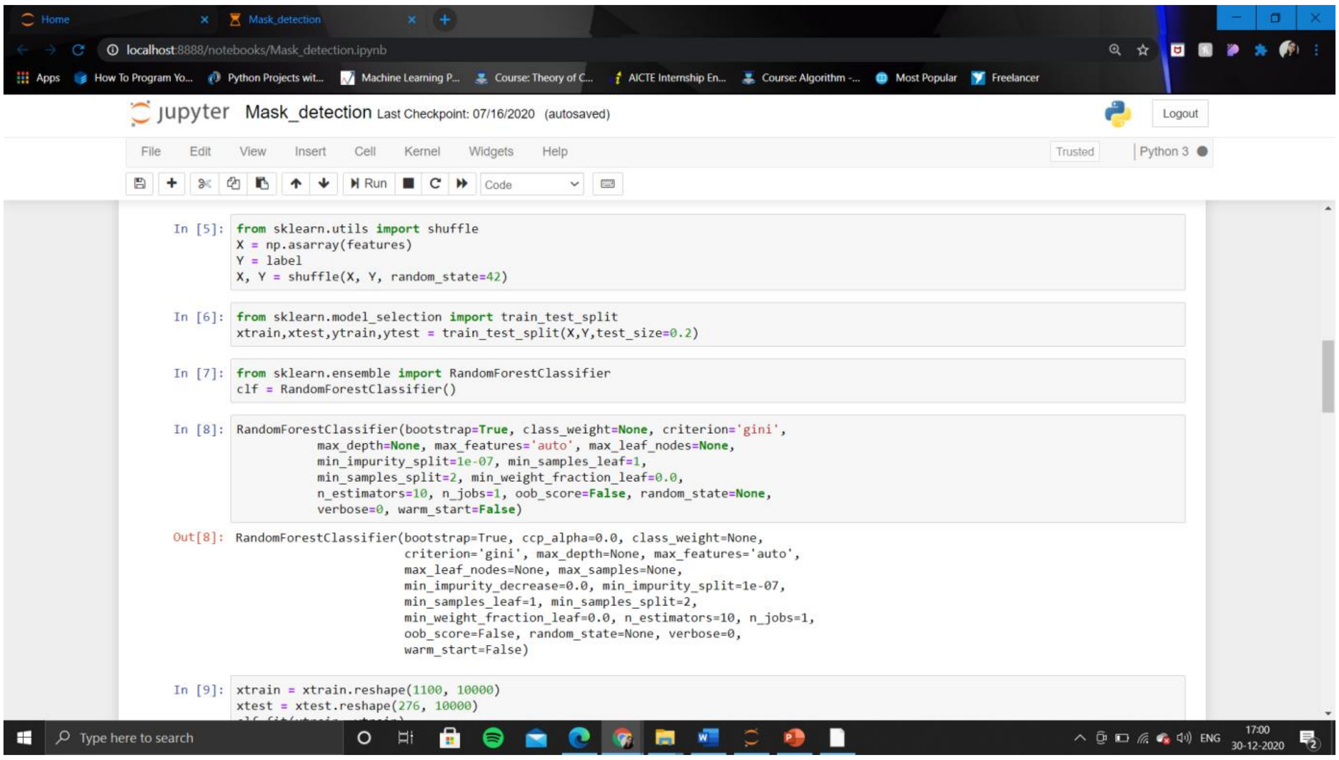
```
In [2]: with_mask = []
        without_mask = []

        mypath_mask = "dataset/with_mask"
        for i in os.listdir(mypath_mask):
            p = os.path.join(mypath_mask,i)
            f = cv2.imread(p,0)
            f = cv2.resize(f,(100,100))
            with_mask.append(list(f))

        mypath_nomask = "dataset/without_mask"
        for i in os.listdir(mypath_nomask):
            p = os.path.join(mypath_nomask,i)
            f = cv2.imread(p,0)
            f = cv2.resize(f,(100,100))
            without_mask.append(list(f))
```

```
In [3]: m_shape = len(with_mask)
        nm_shape = len(without_mask)

        m_array = np.empty(m_shape, dtype=np.int)
        m_array.fill(1)
```



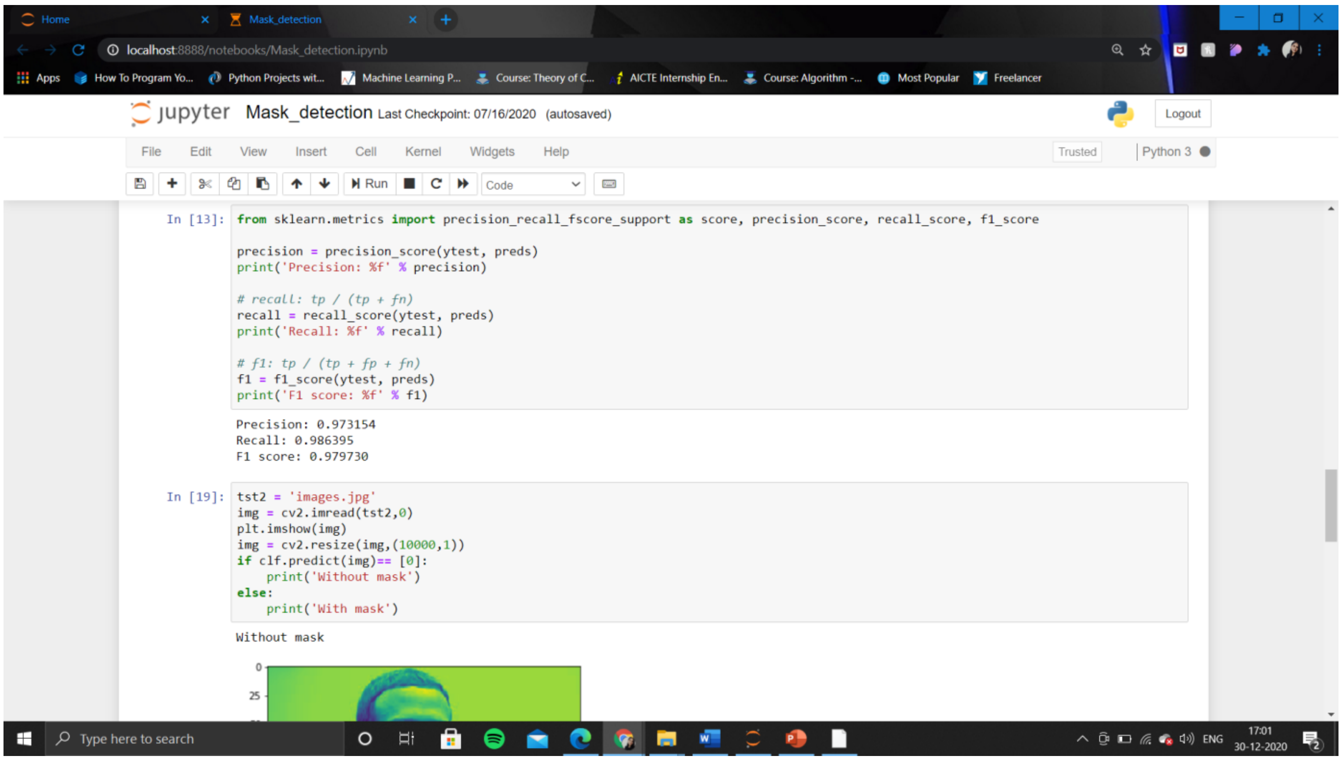
jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
0
20
40
60
80

```
In [3]: m_shape = len(with_mask)
        nm_shape = len(without_mask)

        m_array = np.empty(m_shape, dtype=np.int)
        m_array.fill(1)

        nm_array = np.empty(nm_shape, dtype=np.int)
        nm_array.fill(0)

In [4]: features = with_mask + without_mask
        label = np.concatenate((m_array,nm_array))
        plt.imshow(features[0])

Out[4]: <matplotlib.image.AxesImage at 0x2ce42ce50b8>
```

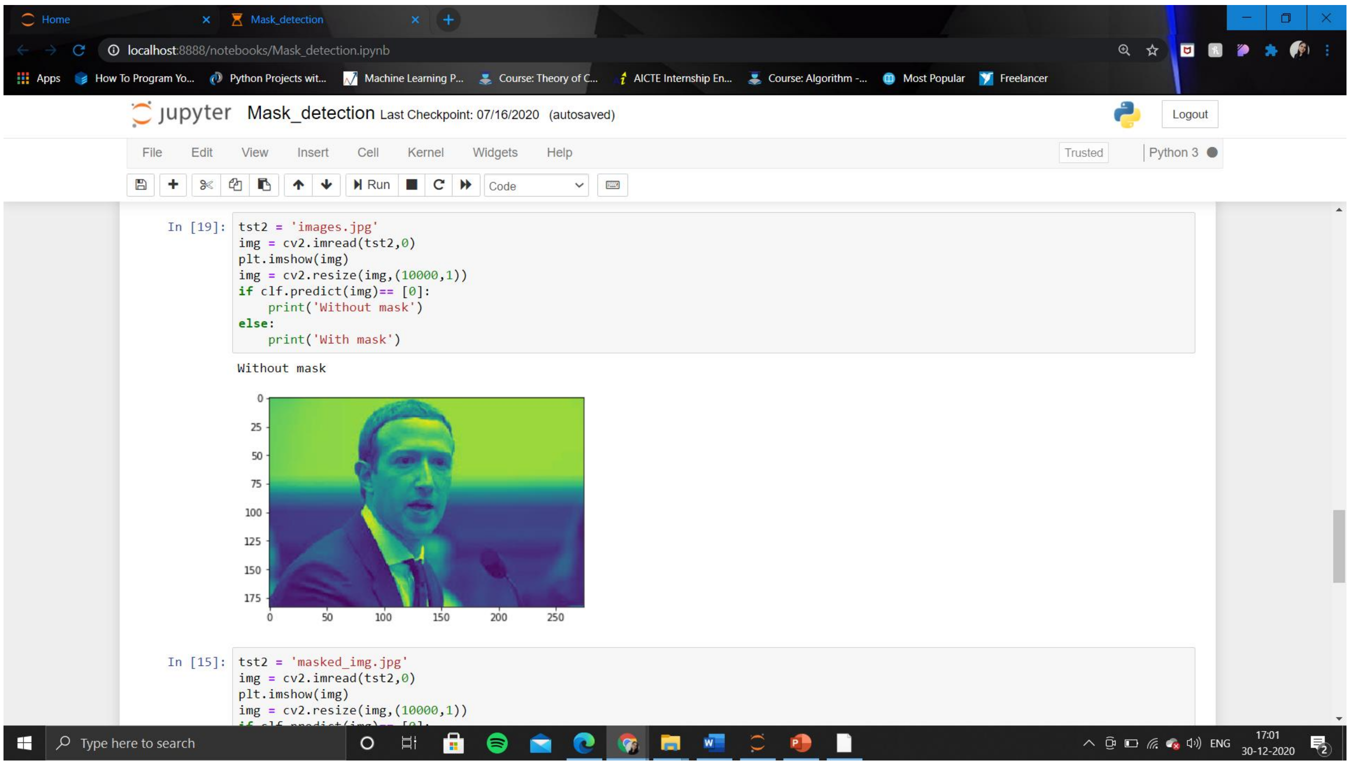


jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3

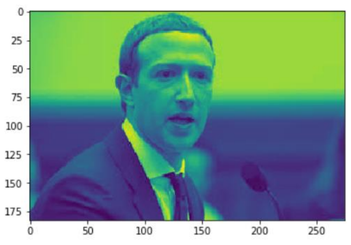
```
In [5]: from sklearn.utils import shuffle
        X = np.asarray(features)
        Y = label
        X, Y = shuffle(X, Y, random_state=42)

In [6]: from sklearn.model_selection import train_test_split
        xtrain,xtest,ytrain,ytest = train_test_split(X,Y,test_size=0.2)

In [7]: from sklearn.ensemble import RandomForestClassifier
        clf = RandomForestClassifier()

In [8]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)

Out[8]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                               criterion='gini', max_depth=None, max_features='auto',
                               max_leaf_nodes=None, max_samples=None,
                               min_impurity_decrease=0.0, min_impurity_split=1e-07,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                               oob_score=False, random_state=None, verbose=0,
                               warm_start=False)

In [9]: xtrain = xtrain.reshape(1100, 10000)
        xtest = xtest.reshape(276, 10000)
```


jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout

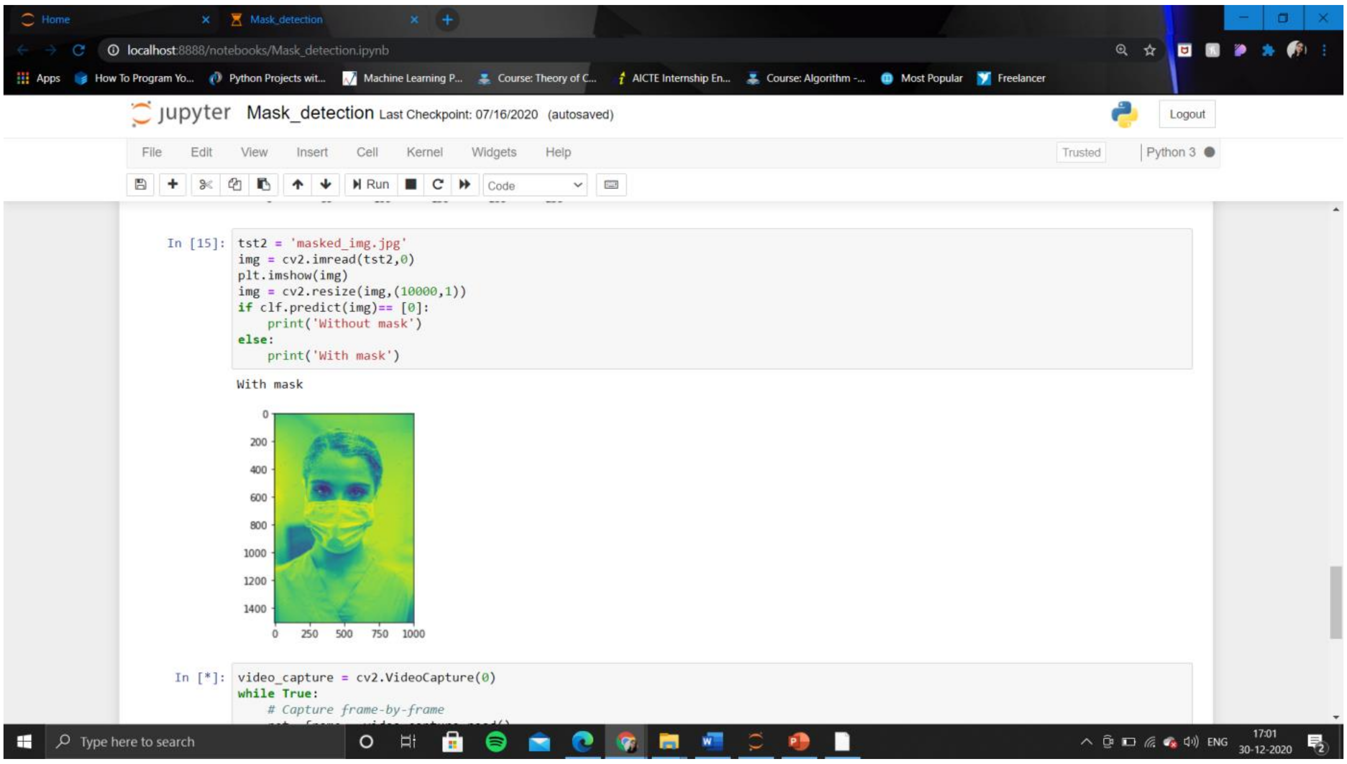
```
In [13]: from sklearn.metrics import precision_recall_fscore_support as score, precision_score, recall_score, f1_score

         precision = precision_score(ytest, preds)
         print('Precision: %f' % precision)

         # recall: tp / (tp + fn)
         recall = recall_score(ytest, preds)
         print('Recall: %f' % recall)

         # f1: tp / (tp + fp + fn)
         f1 = f1_score(ytest, preds)
         print('F1 score: %f' % f1)

         Precision: 0.973154
         Recall: 0.986395
         F1 score: 0.979730

In [19]: tst2 = 'images.jpg'
         img = cv2.imread(tst2,0)
         plt.imshow(img)
         img = cv2.resize(img,(10000,1))
         if clf.predict(img)== [0]:
             print('Without mask')
         else:
             print('With mask')

         Without mask
```


jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout

```
In [19]: tst2 = 'images.jpg'
         img = cv2.imread(tst2,0)
         plt.imshow(img)
         img = cv2.resize(img,(10000,1))
         if clf.predict(img)== [0]:
             print('Without mask')
         else:
             print('With mask')

         Without mask
```

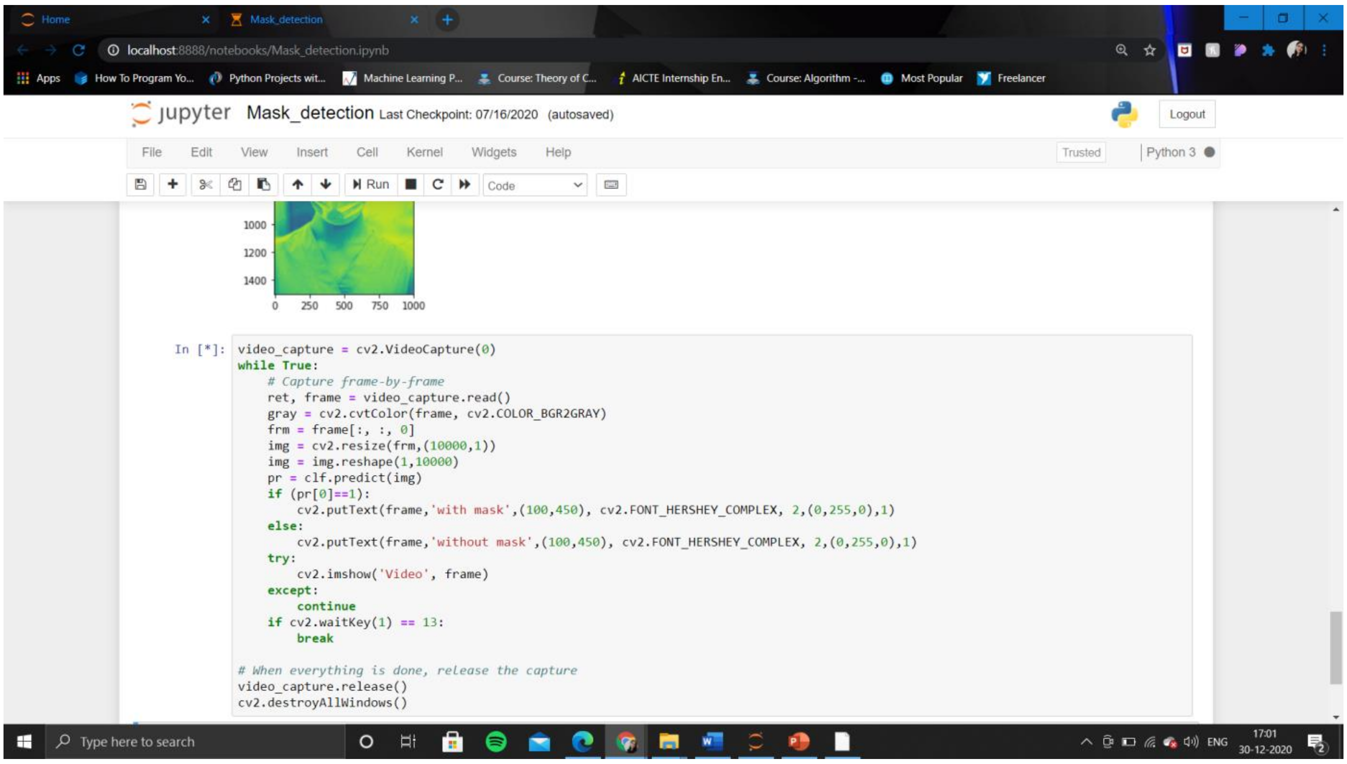


```
In [15]: tst2 = 'masked_img.jpg'
         img = cv2.imread(tst2,0)
         plt.imshow(img)
         img = cv2.resize(img,(10000,1))
```


jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout

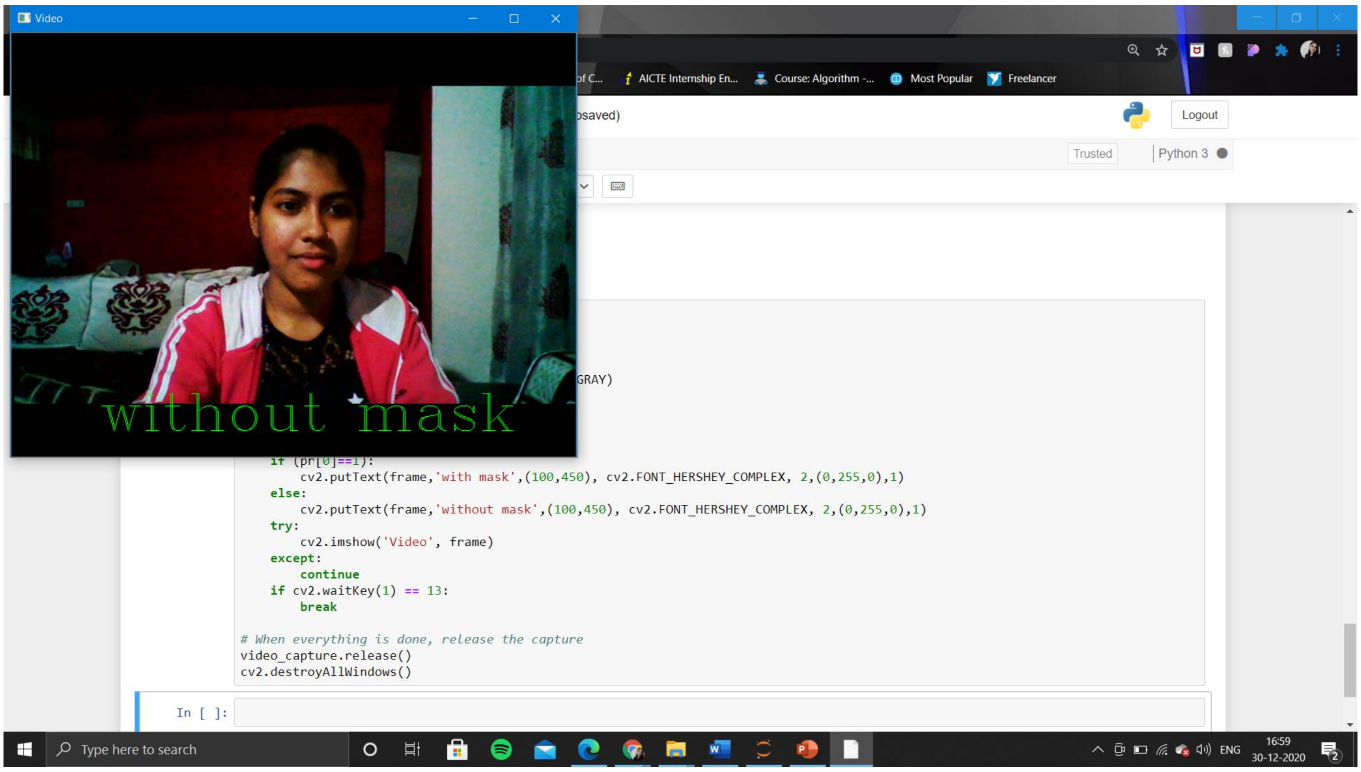
```
In [15]: tst2 = 'masked_img.jpg'
         img = cv2.imread(tst2,0)
         plt.imshow(img)
         img = cv2.resize(img,(10000,1))
         if clf.predict(img)== [0]:
             print('Without mask')
         else:
             print('With mask')

         With mask
```

```
In [*]: video_capture = cv2.VideoCapture(0)
        while True:
            # Capture frame-by-frame
```


jupyter Mask_detection Last Checkpoint: 07/16/2020 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Logout

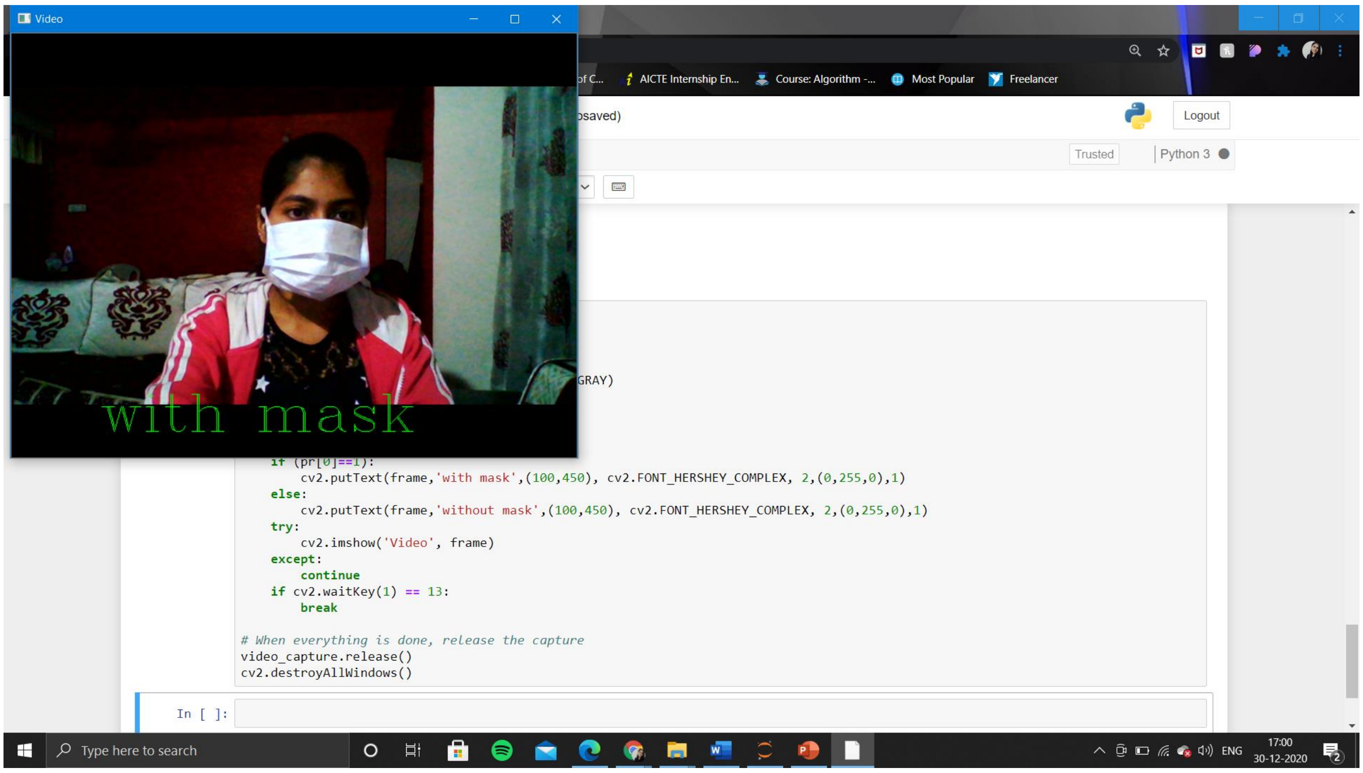
```
In [*]: video_capture = cv2.VideoCapture(0)
        while True:
            # Capture frame-by-frame
            ret, frame = video_capture.read()
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frm = frame[:, :, 0]
            img = cv2.resize(frm,(10000,1))
            img = img.reshape(1,10000)
            pr = clf.predict(img)
            if (pr[0]==1):
                cv2.putText(frame,'with mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
            else:
                cv2.putText(frame,'without mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
            try:
                cv2.imshow('Video', frame)
            except:
                continue
            if cv2.waitKey(1) == 13:
                break

        # When everything is done, release the capture
        video_capture.release()
        cv2.destroyAllWindows()
```

Output



```
GRAY)
    if (pr[0]==1):
        cv2.putText(frame,'with mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    else:
        cv2.putText(frame,'without mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    try:
        cv2.imshow('Video', frame)
    except:
        continue
    if cv2.waitKey(1) == 13:
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

In [ ]:



```
GRAY)
    if (pr[0]==1):
        cv2.putText(frame,'with mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    else:
        cv2.putText(frame,'without mask',(100,450), cv2.FONT_HERSHEY_COMPLEX, 2,(0,255,0),1)
    try:
        cv2.imshow('Video', frame)
    except:
        continue
    if cv2.waitKey(1) == 13:
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

In [ ]:

6. CONCLUSION

To create our face mask detector, we trained a two-class model of people *wearing masks* and people *not wearing masks.* The main aim of the project is satisfied i.e., the model is capable to detect faces with mask and without mask and works well with the dataset used. The model may not be successful every time but it gives correct prediction with a good accuracy score. The machine learning model coded using python is fit enough to classify images into two classes 'with mask' and 'without mask'. Using random forest classifier provides better approach than many other classifiers and also reduces the possibility of overfitting. The code works fine with the images other than that in the dataset. This is a software project but for further applications in real world scenario it can be integrated with certain hardware equipment and can be installed in various public places for surveillance of people wearing mask or not.

7. FUTURE ENHANCEMENTS

As you can see from the results sections above, our face mask detector is working quite well despite:

1. Having limited training data
2. The with_mask class being artificially generated (see the *"How was our face mask dataset created?"* section above).

To improve our face mask detection model further, you should gather *actual images* (rather than artificially generated images) of people wearing masks.

While our artificial dataset worked well in this case, there's no substitute for the real thing.

Secondly, you should also gather images of faces that may "confuse" our classifier into thinking the person is wearing a mask when in fact they are not — potential examples include shirts wrapped around faces, bandana over the mouth, etc.

All of these are examples of something that *could* be confused as a face mask by our face mask detector.

Finally, you should consider training a dedicated two-class *object detector* rather than a simple image classifier.

Our current method of detecting whether a person is wearing a mask or not is a two-step process:

1. Step #1: Perform face detection
2. Step #2: Apply our face mask detector to each face

The *problem* with this approach is that a face mask, by definition, obscures part of the face. If enough of the face is obscured, the face cannot be detected, *and therefore, the face mask detector will not be applied.*

To circumvent that issue, you should train a two-class object detector that consists of a with_mask class and without_mask class.

Combining an object detector with a dedicated with_mask class will allow improvement of the model in two respects.

First, the object detector will be able to naturally detect people wearing masks that otherwise would have been impossible for the face detector to detect due to too much of the face being obscured.

Secondly, this approach reduces our computer vision pipeline to a single step — rather than applying face detection and *then* our face mask detector model, all we need to do is apply the object detector to give us bounding boxes for people both `with_mask` and `without_mask` in a single forward pass of the network.

Not only is such a method more computationally efficient, it’s also more “elegant” and end-to-end.

BIBLIOGRAPHY

https://www.who.int/news-room/q-a-detail/coronavirus-disease-covid-19-masks

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

https://link.springer.com/chapter/10.1007/978-3-540-73007-1_85

https://en.wikipedia.org/wiki/Computer_vision#:~:text=Computer%20vision%20is%20an%20interdisciplinary,human%20visual%20system%20can%20do.

https://scikit-learn.org/stable/modules/model_evaluation.html