

# Project – Face Mask Detection

## :: Project Details ::

- Authors :  
    Varshith      (160118737055)  
    Vishwahith   (160118737060)
- Date of completion :  
    Nov 24, 2020 - 22:00 IST

## :: Introduction ::

**Face Mask Detection**, a computer-based project built with Machine Learning constructs.

This project is one of the specializations of the parent project namely "*Face Detection*".

## :: History of Face-Detection ::

The world believes that Woodrow Wilson Bledsoe was the father of facial recognition. In the 1960s, Bledsoe created a system that could organize faces' photos by hand using the RAND tablet. The tablet is a device people could use to enter vertical and horizontal coordinates on a grid with the help of a stylus that

released electromagnetic pulses. People used that system to manually record the coordinate areas of facial features like eyes, nose, mouth, and hairline.

The manually recorded metrics could be later saved within a database. And when the new photograph of an individual was entered into the system, it was able to get the most closely resembled image via database. During this period, face recognition was untouched by technology and computer processing power. Still, it was the first and foremost step taken by Bledsoe to prove that face recognition was a practical biometric.

Sirovich and Kirby started using linear algebra to the issue of facial recognition in 1988. The approach they used was called the Eigenface approach. The rendering began as a search for low-dimensional facial images representation. The team was able to prove that feature analysis on collected pictures in the database could form a set of basic features.

:: What is Face Detection/Recognition ? ::

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene. There are many algorithms to complete this task. In terms of speed, HoG seems to be the fastest **algorithm**, followed by Haar Cascade classifier and CNNs. However, CNNs in Dlib tend to be the most accurate **algorithm**. HoG perform pretty well but have some issues identifying small **faces**. HaarCascade Classifiers perform around as **good** as HoG overall.

:: Algorithms ::

There are enormous number of algorithms among which these are the robust and scalable algorithms:

- [Eigenfaces](#) (1991)
- [Local Binary Patterns Histograms \(LBPH\)](#) (1996)
- [Fisherfaces](#) (1997)
- [Scale Invariant Feature Transform \(SIFT\)](#) (1999)
- [Speed Up Robust Features \(SURF\)](#) (2006)

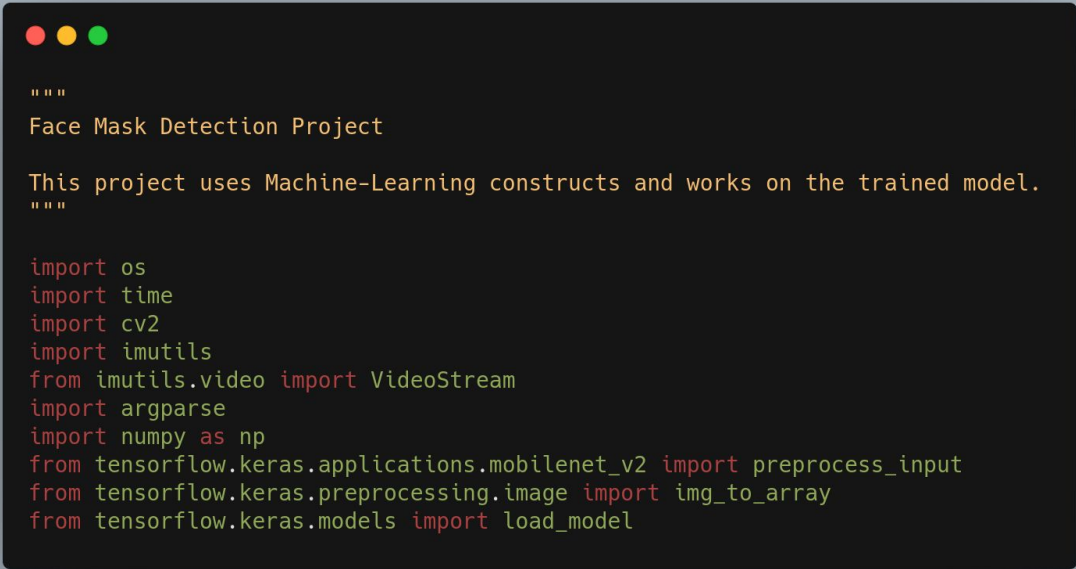
## :: Explanation on Code ::

The code for this project is dominantly written in *"Python-3.8.6"*.

To understand the code clearly, it is explained in smaller snippets (or units) as below.

The code-base for this project is of size 1.8 GiB. This is large because of the dataset used to train the model. The *main.py* is our application which uses the trained model and responds as desired.

Initially the dependent modules were imported as referred in PEP8 style-guide.



```
"""
Face Mask Detection Project

This project uses Machine-Learning constructs and works on the trained model.
"""

import os
import time
import cv2
import imutils
from imutils.video import VideoStream
import argparse
import numpy as np
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
```

As depicted above the dependencies of the main code are:

- tensorflow>=1.15.2
- keras==2.3.1
- imutils==0.5.3
- numpy==1.18.2
- opencv-python==4.2.0.\*
- matplotlib==3.2.1
- argparse==1.1
- scipy==1.4.1
- scikit-learn==0.23.1
- pillow==7.2.0
- streamlit==0.65.2

Also the python interpreter used during the development of this application as mentioned above is

`"Python-3.8.6"`.

Right then, we have our responsible function `detect_and_predict_mask` which takes three parameters named as `frame`, `faceNet` and `maskNet`.

This function later returns a tuple, which consists of two elements. These two elements are the coordinates of the face and predictions on those recognized faces, whether they wear masks or not. The elements in this tuple are named as: `(locs, preds)`. These two were later used in the code for the retrieval of the data produced by `detect_and_predict_mask`.

```

def detect_and_predict_mask(frame, faceNet, maskNet):
    """Returns a tuple of 2 elements, which are
    face-coordinates and predictions"""
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
                                  (104.0, 177.0, 123.0))

    faceNet.setInput(blob)
    detections = faceNet.forward()

    """Initialize our list of faces, their corresponding locations,
    and the list of predictions from our face mask network"""
    faces = []
    locs = []
    preds = []

    for i in range(0, detections.shape[2]):
        """Extraction of probability associated with
        the detection"""
        confidence = detections[0, 0, i, 2]

        # Filtration
        if confidence > args["confidence"]:
            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # Maintaining box dimensions
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)

            faces.append(face)
            locs.append((startX, startY, endX, endY))

    # only make a predictions if at least one face was detected
    if len(faces) > 0:
        # for faster inference we'll make batch predictions on *all*
        # faces at the same time rather than one-by-one predictions
        # in the above `for` loop
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    return (locs, preds)

```

Right then, we propose the required arguments which have default options, which can be customized during the execution of the code. That is sited in the code-snippet below.

```
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
                default="face_detector",
                help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
                default="mask_detector.model",
                help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
                help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
                                "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

So here, we also begin the video stream which is handled by the `imutils` module. We also do some logging for serving the user with all the details. We load our machine-learning trained model, which is helpful for the deduction of the faces consisting of masks or not.



```

# Looping over the frames from the video stream
while True:
    """Grabbing the frame from the threaded video stream and resize it
    to have a maximum width of 1000 pixels"""
    frame = vs.read()
    frame = imutils.resize(frame, width=1000)

    # Detect and predict the face, whether it consists of mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # Loop over the detected face locations and their surroundings
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # Color label determination
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        # Including the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        # Setting output-frame
        cv2.putText(frame, label, (startX, startY - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    # Display the output-frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # End the session on pressing `q`
    if key == ord("q"):
        vs.stream.release()
        #cv2.destroyAllWindows()
        print("[EXIT] End of the session.")
        break

```

Finally, we have this ever-lasting loop for live-streaming the deductions of our ML model. If the



user press the letter **q** from his/her keyboard the sequence terminates and the code-execution exits.

Hence, this is the a thorough explanation of our heart of the application `main.py`.

## :: Execution Environment ::

This project is developed on `Linux fedora 5.9.8-200.fc33.x86_64`. Therefore, to run the code successfully on other machines, the required dependencies are clearly published under `etc` directory.

No manual addition of ENVIRONMENT VARIABLES are needed for this application to be executed.

## :: Experiencing the runtime ::

# To experience the runtime of this follow these *first-steps*:

- Move the tarball/zip-file of this app to Downloads directory.

```
$ cd $HOME/Downloads
$ unzip Face-Mask-Detection.zip

$ pip install -r etc/requirements.txt

$ python main.py
```

Therefore, you'll start experiencing the runtime of this application.

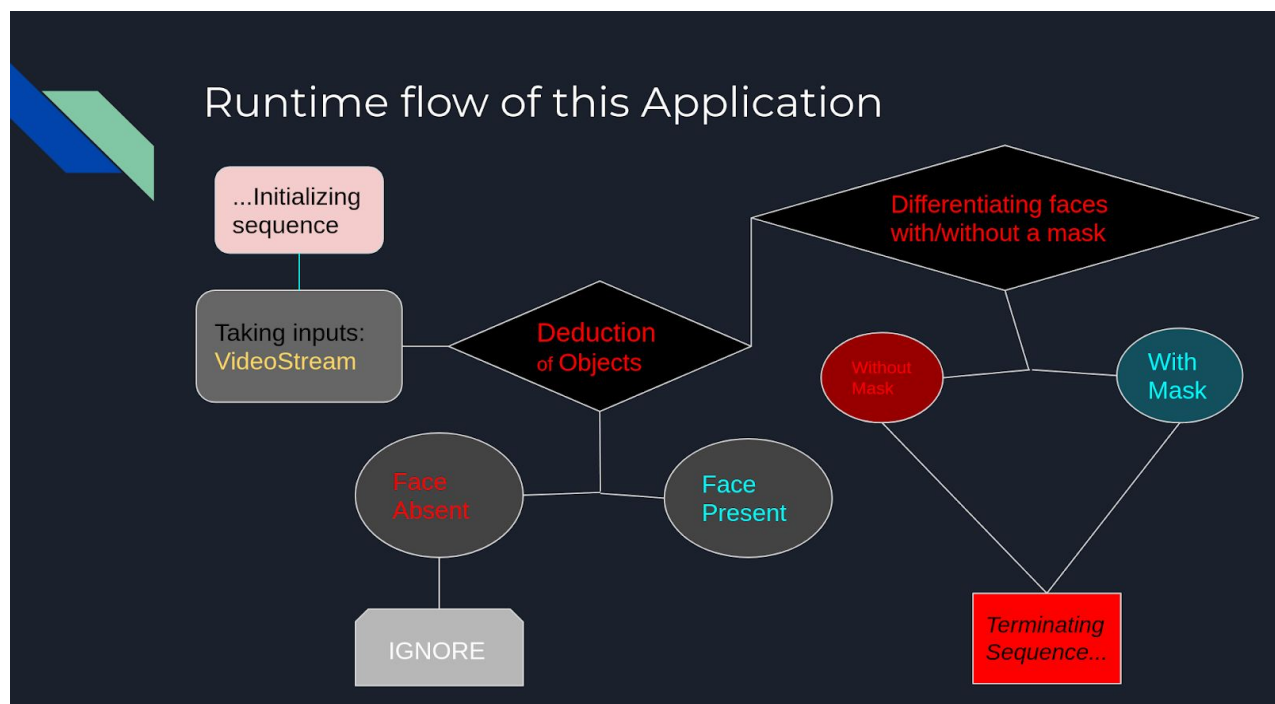
If you want to quit the running app, simply press **q**.

**Note:** It is advised to clone this app on a Linux Host with Python3.8 interpreter.

:: Conclusion ::

Therefore, the application has been designed successfully in the desired architecture. The documentation is pretty straight forward to follow and execute the application for experiencing.

The execution flow of this application is visualized in below flow chart.



:: END ::