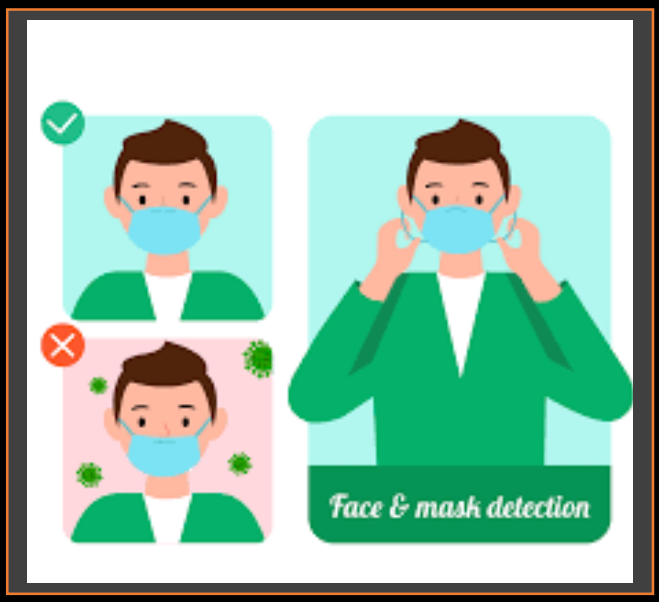


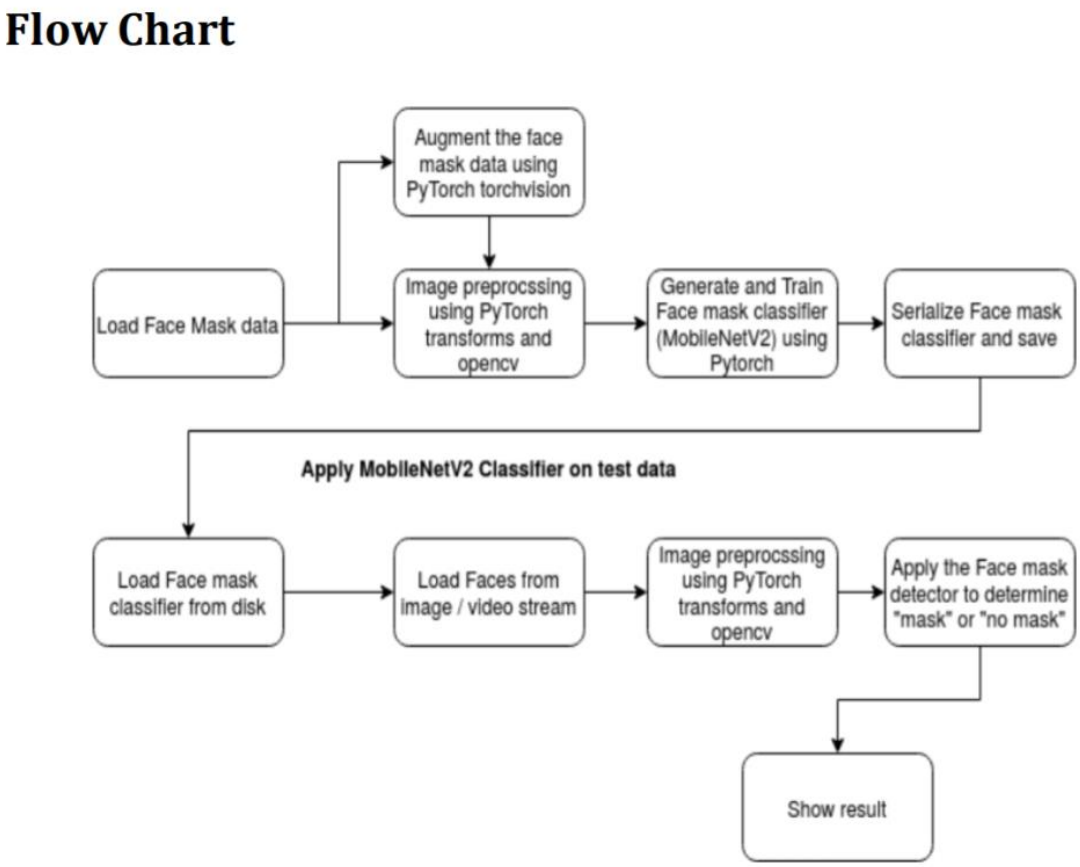
FACE MASK DETECTION

BY : -
1) Aditya Mantri – 20184151
2) Abhinash Murmu – 20184186
3) Beri Prashanth – 20184173
4) Gaurav Bansal - 20184155

OBJECTIVE

Here we introduce a mask face detection model that is based on computer vision and deep learning. The proposed model can be integrated with surveillance cameras to impede the COVID-19 transmission by allowing the detection of people who are wearing masks not wearing face masks. The model is integration between deep learning and classical machine learning techniques with openCV, tensor flow and keras.

FLOWCHART –



EXECUTION -

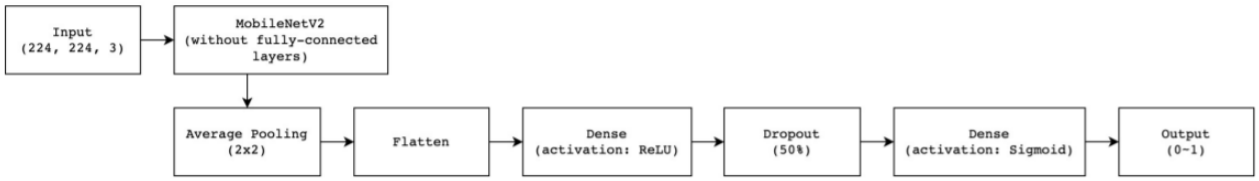


Figure 3: Process Flow diagram of the model.

WHY THIS PROJECT?

In 2020, the rapid spreading of COVID-19 has forced the World Health Organization to declare COVID- 19 as a global pandemic. More than five million cases were infected by COVID-19 in less than 6 months across 188 countries. The virus spreads through close contact and in crowded and overcrowded areas. The corona virus epidemic has given rise to an extraordinary degree of worldwide scientific cooperation. Artificial Intelligence (AI) based on Machine learning and Deep Learning can help to fight Covid-19 in many ways. Machine learning allows researchers and clinicians evaluate vast quantities of data to forecast the distribution of COVID-19,to serve as an early warning mechanism for potential pandemics, and to classify vulnerable population.

People are forced by laws to wear face masks in public in many countries.These rules and laws were developed as an action to the exponential growth in cases and deaths in many areas. However, the process of monitoring large groups of people is becoming more difficult. The monitoring process involves the detection of anyone who is not wearing a face mask.

STEP 1)

Install below Dependencies on Anaconda Prompt

```
tensorflow>=1.15.2
keras==2.3.1
imutils==0.5.3
numpy==1.18.2
opencv-python==4.3.9
matplotlib==3.2.1
scipy==1.4.1
```

STEP 2)

PRE PROCESSING THE DATA SET-

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r"C:\Mask Detection\CODE\Face-Mask-Detection-master\dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

```
# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images


data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
```

```
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
```

STEP 3)

TRAINING THE MODEL –

Train Deep learning model (MobileNetV2)

At the training time, for each pixel, we compare the default bounding boxes having different sizes and aspect ratios with ground truth boxes and finally use Intersection over Union (IoU) method to select the best matching box. IoU evaluates how much part of our predicted box match with the ground reality. The values range from 0 to 1 and increasing values of IoU determine the accuracies in the prediction; the best value being the highest value of IoU. The equation and pictorial description of IoU is given as:

$$IoU(B_1, B_2) = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$

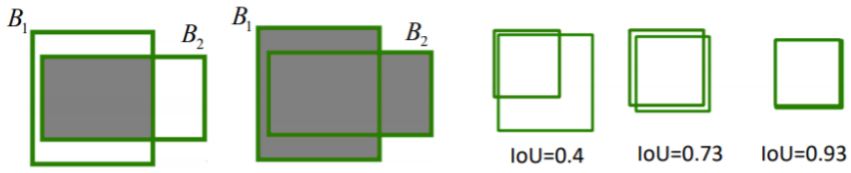


Figure 4: Pictorial representation of IoU.

```
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
```

```
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit(
```

```
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

STEP 4)

TESTING –

We tried using three different base models for detecting ‘mask’ or ‘no mask’. The exercise was done to find the best fit model in our scenario. The evaluation process consists of first looking at the classification report which gives us insight towards precision, recall and F1 score. The equations of these three metrics are as follows:

$$Precision = \frac{True\ Positives}{Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{Positives + False\ Negatives}$$

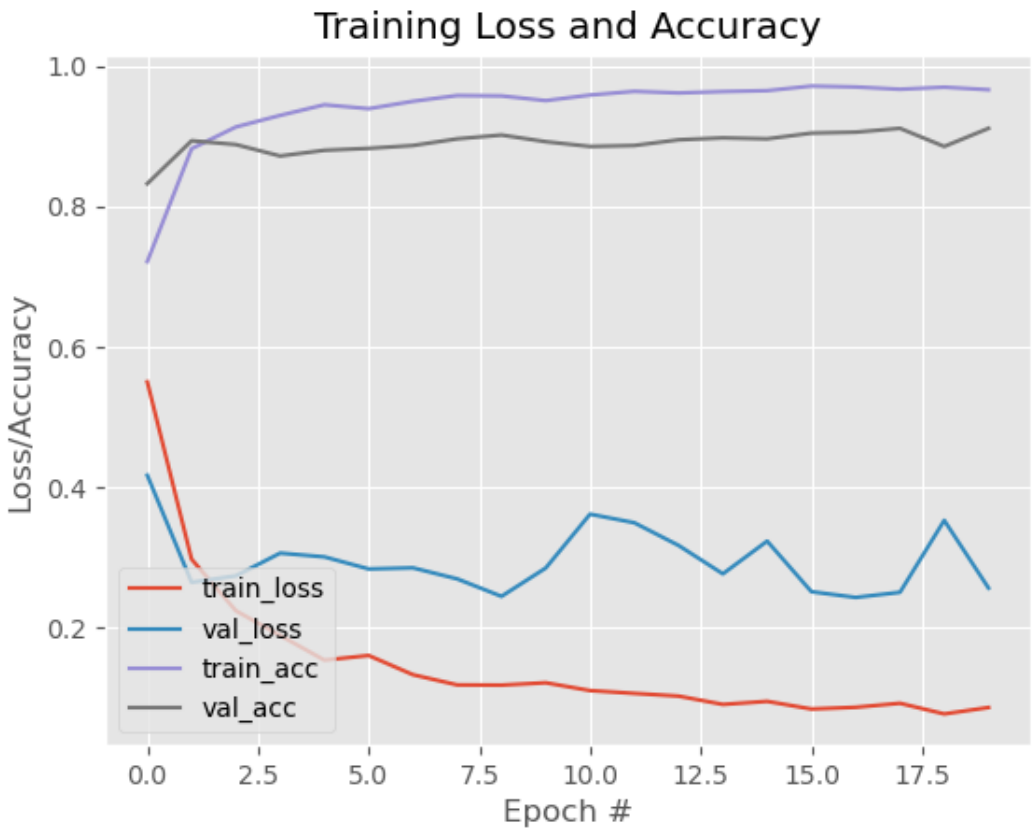
$$Accuracy = \frac{True\ Positives + True\ Negatives}{Positives + Negatives}$$

Using these three metrics, we can conclude which model is performing most efficiently. The second part consists of plotting the train loss, validation loss, train accuracy and validation accuracy which also proves helpful in choosing a final model.

STEP 5)

RUNNING THE PREPROCESSING AND TRAINING ON THE DATA SET -

```
[INFO] evaluating network...
classification report:

              precision    recall  f1-score   support

   with_mask       0.98      0.99      0.99       384
without_mask       0.99      0.98      0.99       386

    accuracy                           0.99       770
   macro avg       0.99      0.99      0.99       770
weighted avg       0.99      0.99      0.99       770
```



- We get the ***MODEL.h5*** file now after the execution.

STEP 6)

USING MODEL AND GENERATING VIDEO OUTPUT

```
# import the necessary packages
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
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
```

```
import os
```

```
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)

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

```
# loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the confidence is
        # greater than the minimum confidence
        if confidence > 0.5:
            # 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")

            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (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)

            # add the face and bounding boxes to their respective
            # lists
            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 a 2-tuple of the face locations and their corresponding
    # locations
    return (locs, preds)
```

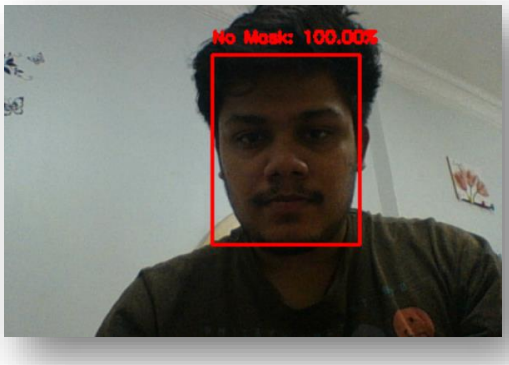
```
# load our serialized face detector model from disk
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model("mask_detector.model")

# initialize the video stream print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
```

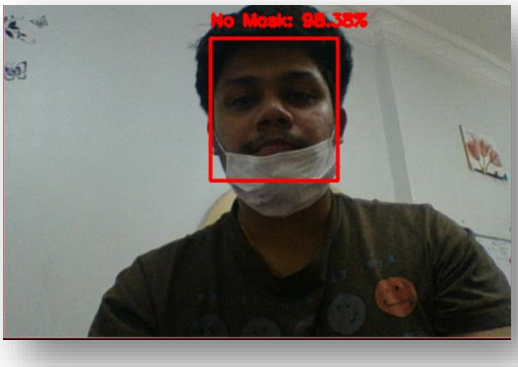
```
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
```

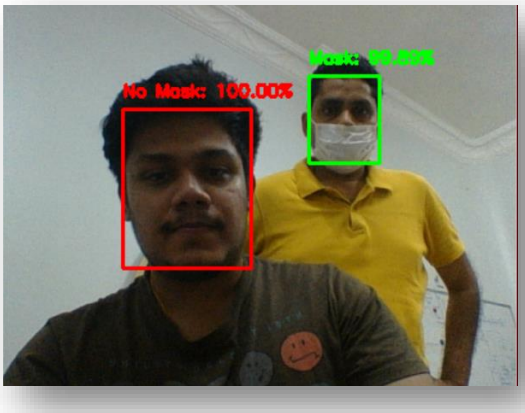
```
    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

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

        # display the label and bounding box rectangle on the 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)

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

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break
# do a bit of cleanup
cv2.destroyAllWindows()
```

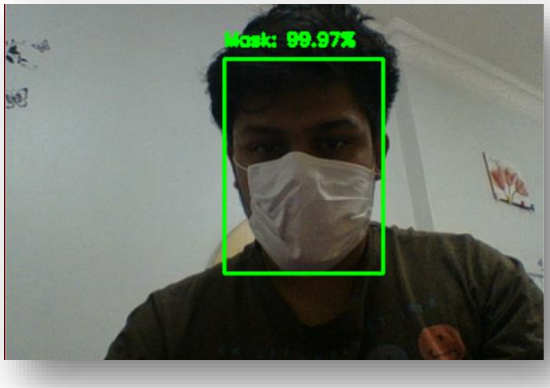
STEP 7)

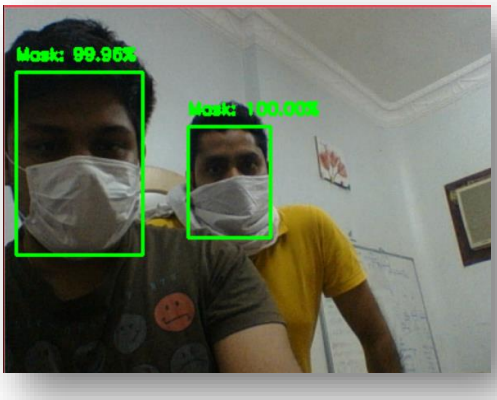
FINAL OUTPUT AND END RESULT-











The average execution time of the system for processing one image frame was 0.13 s, which showed that our system could detect face masks in real-time and was of value in practical application

STEP 7)

INTEGRATION WITH WEB DEVELOPMENT

Index.html

```
<!DOCTYPE html>

<html>
<head>
    <title></title>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/blazeface"></script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5
.2/css/bootstrap.min.css" integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z" crossorigin="an
onymous">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.0/css/
all.css" integrity="sha384-
lZN37f5QGtY3VHgisS14W3ExzMWZxybE1SJSEsQp9S+oqd12jhcu+A56Ebc1zFSJ" crossorigin="an
onymous">
</head>
<style>

body {
    background-
image: url('https://americalatinagenera.org/newsite/images/2019/Cambios-
sitio/covid-19-fondo.jpg');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: cover;
}

.bg-dark{
    display: inline-block;
}

</style>
<body >
    <div class="container p-0" >
        <header >
          <nav class="navbar navbar-dark bg-dark rounded">
            <a class="navbar-brand">
                <i class="fas fa-camera"></i>
```

```
                <strong>Masks Detection</strong>
            </a>
          </nav>
        </header>
    </div>


    <div class="container ">
        <div class="row" style="height:480px">
            <video id="video" playsinline class="border " style="margin:auto;disp
lay:inline-block;"></video>
            <canvas id="output" class="canvas-
output" style="margin:auto;position:relative;top:-480px;left:10px"></canvas>
        </div>
    </div>
</body>
<script>
    //Coded by oh yicong, visit my youtube channel for more programming tutorials
 :)
    var model, mask_model, ctx, videoWidth, videoHeight, canvas;
    const video = document.getElementById('video');
    const state = {
      backend: 'webgl'
    };
    async function setupCamera() {
        const stream = await navigator.mediaDevices.getUserMedia({
            'audio': false,
            'video': { facingMode: 'user' },
        });
        video.srcObject = stream;
        return new Promise((resolve) => {
            video.onloadedmetadata = () => {
              resolve(video);
            };
        });
    }

    const renderPrediction = async () => {
        tf.engine().startScope()
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        //estimatefaces model takes in 4 parameter (1) video, returnTensors, flip
Horizontal, and annotateBoxes
        const predictions = await model.estimateFaces(video, true,false,false);
        const offset = tf.scalar(127.5);
        //check if prediction length is more than 0
```

```
        if (predictions.length > 0) {
            //clear context

            for (let i = 0; i < predictions.length; i++) {
                var text=""
                var start = predictions[i].topLeft.arraySync();
                var end = predictions[i].bottomRight.arraySync();
                var size = [end[0] - start[0], end[1] - start[1]];
                if(videoWidth<end[0] && videoHeight<end[0]){
                    console.log("image out of frame")
                    continue
                }
                var inputImage = tf.browser.fromPixels(video).toFloat()
                inputImage = inputImage.sub(offset).div(offset);
                inputImage=inputImage.slice([parseInt(start[1]),parseInt(start[0]
),0],[parseInt(size[1]),parseInt(size[0]),3])
                inputImage=inputImage.resizeBilinear([224,224]).reshape([1,224,22
4,3])
                result=mask_model.predict(inputImage).dataSync()
                result= Array.from(result)
                ctx.beginPath()
                if (result[1]>result[0]){
                    //no mask on
                    ctx.strokeStyle="red"
                    ctx.fillStyle = "red";
                    text = "No Mask: "+(result[1]*100).toPrecision(3).toString()+
"%";
                }else{
                    //mask on
                    ctx.strokeStyle="green"
                    ctx.fillStyle = "green";
                    text = "Mask: "+(result[0]*100).toPrecision(3).toString()+"%"
;
                }
                ctx.lineWidth = "4"
                ctx.rect(start[0], start[1],size[0], size[1])
                ctx.stroke()
                ctx.font = "bold 15pt sans-serif";
                ctx.fillText(text,start[0]+5,start[1]+20)
            }
        }
        //update frame
        requestAnimationFrame(renderPrediction);
        tf.engine().endScope()
    };
```

```
    const setupPage = async () => {
        await tf.setBackend(state.backend);
        await setupCamera();
        video.play();

        videoWidth = video.videoWidth;
        videoHeight = video.videoHeight;
        video.width = videoWidth;
        video.height = videoHeight;

        canvas = document.getElementById('output');
        canvas.width = videoWidth;
        canvas.height = videoHeight;
        ctx = canvas.getContext('2d');
        ctx.fillStyle = "rgba(255, 0, 0, 0.5)";

        model = await blazeface.load();

        mask_model = await tf.loadLayersModel('../static/models/model.json');

       renderPrediction();
    };

    setupPage();

</script>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-
DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="an
onymous"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js
" integrity="sha384-
9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossorigin="an
onymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.
js" integrity="sha384-
B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV" crossorigin="an
onymous"></script>

</html>
```

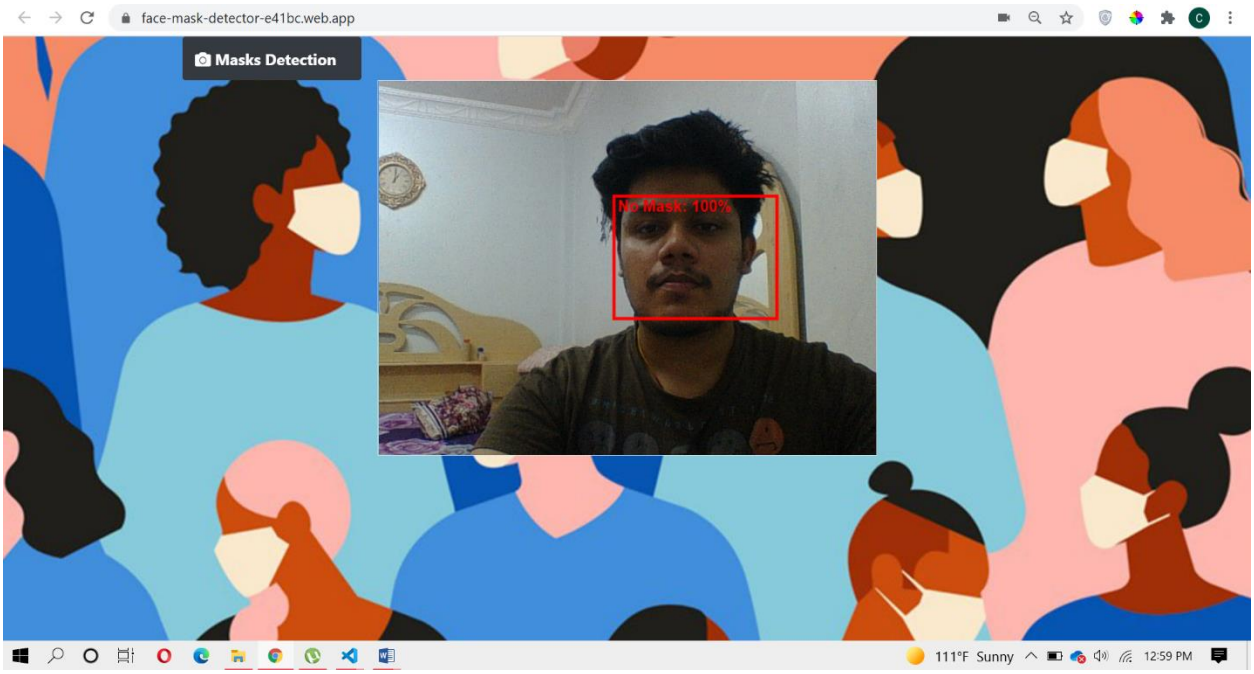
Web_main.py

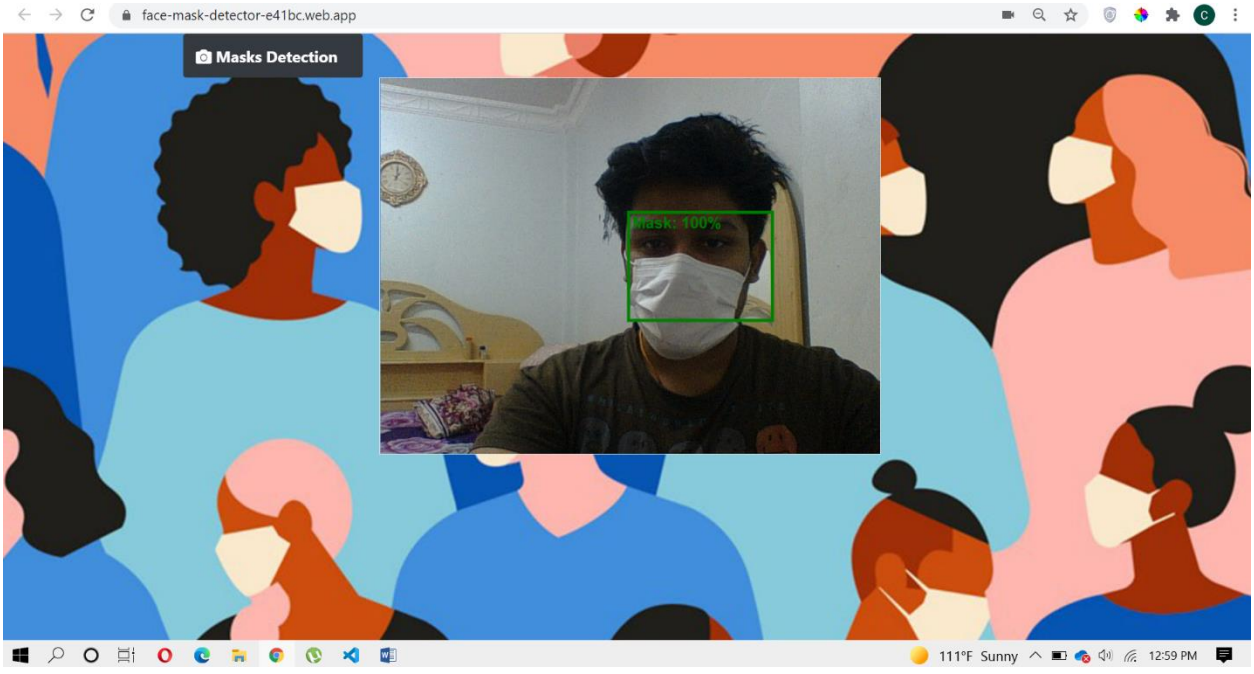
```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  8 08:49:23 2020

@author: OHyic
"""

#import flasks libraries
from flask import Flask, render_template, request, jsonify, url_for, redirect, fl
ash, Response, send_file
from flask_socketio import SocketIO, emit
import webbrowser
from threading import Timer

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)


@app.route('/')
def index():
    return render_template('index.html')


if __name__ == '__main__':
    Timer(1,lambda: webbrowser.open_new("http://127.0.0.1:5000/")).start()
    socketio.run(app)
```

STEP 8)

Deployment to web using Firebase hosting facility

Link : https://face-mask-detector-e41bc.web.app





FINAL CONCLUSION –

As the technology are blooming with emerging trends the availability so we have novel face mask detector which can possibly contribute to public healthcare. The architecture consists of MobileNet as the backbone it can be used for high and low computation scenarios. In order to extract more robust features, we utilize transfer learning to adoptweights from a similar task face detection, which is trained on a very large dataset. We used OpenCV, tensor flow, keras , Pytorch and CNN to detect whether people were wearing face masks or not. The models were tested with images and real-time video streams. The accuracy of the model is achieved and, the optimization of the model is a continuous process and we are building a highly accurate solution by tuning the hyper parameters. This specific model could be used as a use case for edge analytics. Furthermore, the proposed method achieves state-of-the-art results on a public face mask dataset. By the development of face mask detection, we can detect if the person is wearing a face mask and allow their entry would be of great help to the society.

REFERENCES –

1) MobileNET https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet/MobileNet
2) Model Plotting https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
3) Relu https://datascience.eu/machine-learning/relu-activation-function/
4) Softmax https://dl.acm.org/doi/10.5555/3298023.3298043
5) Handling OverFitting https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
6) Analysis of augmented dataset https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0