PLOT NO.98 SECTOR-3, AIROLI, NAVI MUMBAI

ACADAMIC YEAR 2018-19

DATTA MEGHE COLLEGE OF ENGINEERING

AIROLI, NAVI MUMBAI

CERTIFICATE

This is to certify that the project entitled "**Face Mask Detection**" is bona fide work of "**SanketDhake and BhargavPhadke** " submitted to the University of Mumbai in partial fulfilment of the requirement for the award of **TE** in **"Computer Engineering"**.

Prof. ManojPatil	Prof. A. P. Pande	Dr.S.D.Sawarkar
Project Guide	Head of the Department	Principal

A MINI PROJECT REPORT

ON

"FACE MASK DETECTION"

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF COMPUTER ENGINEERING

BY

SanketDhake(Roll No 5)

BhargavPhadke(Roll No 34)

UNDER GUIDANCE OF

PROF: Manoj Patil

UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER ENGINEERING

DATTA MEGHE COLLEGE OF ENGINEERING

PLOT NO.98 SECTOR-3, AIROLI, NAVI MUMBAI

DATTA MEGHE COLLEGE OF ENGINEERING

AIROLI, NAVI MUMBAI

MINI PROJECT APPROVAL

This project report entitled "**Face Mask Detection**" of the students"**Sanket Dhake and Bhargav Phadke"**approved for the degree of **Computer Engineering.**

Internal Examiner	External Examiner
Date:	Date:
Place:	Place:

ACKNOWLEDGEMENT

Motivation and guidance are the keys towards success. I would like to extend my thanks to all the sources of motivation.

We would like to grab this opportunity to thank **Dr. S. D. Sawarkar, Principal** for encouragement and support he has given for our project.

We express our deep gratitude to **Prof. A. P. Pande, Head of the Department** who has been the constant driving force behind the completion of this project.

We wish to express our heartfelt appreciation and deep sense of gratitude to my project guide **Prof. Manoj Patil** for his/her encouragement, invaluable support, timely help, lucid suggestions and excellent guidance which helped us to understand and achieve the project goal. His/Her concrete directions and critical views have greatly helped us in successful completion of this work.

We extend our sincere appreciation to allProfessors for their valuable inside and tip during the designing of the project. Their contributions have been valuable in so many ways that we find it difficult to acknowledge of them individually.

We are also thankful to all those who helped us directly or indirectly in completion of this work.

Place: Airoli DhakeSanket

Date: 24/4/2021 BhargavPhadke

INDEX

SR.NO.	TITLE	PAGE NO.
1	INTRODUCTION	6
2	REQUIREMENT ANALYSIS	8
3	PROJECT DESIGN(flow chart)	9
4	TECHNOLOGIES USED	10
5	RESULTS AND DISCUSSIONS	13
6	CONCLUSIONS AND FUTURE SCOPE	24
7	REFERENCES	25

1. INTRODUCTION:

The World Health Organization (WHO) reports suggest that the two main routes of transmission of the COVID-19 virus are respiratory droplets and physical contact.

Respiratory droplets are generated when an infected person coughs or sneezes. Any person in close contact (within 1 m) with someone who has respiratory symptoms (coughing, sneezing) is at risk of being exposed to potentially infective respiratory droplets.

Droplets may also land on surfaces where the virus could remain viable; thus, the immediate environment of an infected individual can serve as a source of transmission (contact transmission). Wearing a medical mask is one of the prevention measures that can limit the spread of certain respiratory viral diseases, including COVID-19.

In this study, medical masks are defined as surgical or procedure masks that are flat or pleated (some are shaped like cups); they are affixed to the head with straps. They are tested for balanced high filtration, adequate breathability and optionally, fluid penetration resistance. The study analyses a set of video streams/images to identify people who are compliant with the government rule of wearing medical masks. This could help the government to take appropriate action against people who are non-compliant.

1.1. Problem Statement:

Create a model which can predict whether person is wearing mask or not by using the MobileNet and OpenCV/ Keras library.

1.2. Approach

1. Train Deep learning model for mask detection (MobileNetV2)
2. Import face detection model.

3. Apply face detector and mask detector over images / live video stream

1.3. System Overview:

Two-phase COVID-19 face mask detector

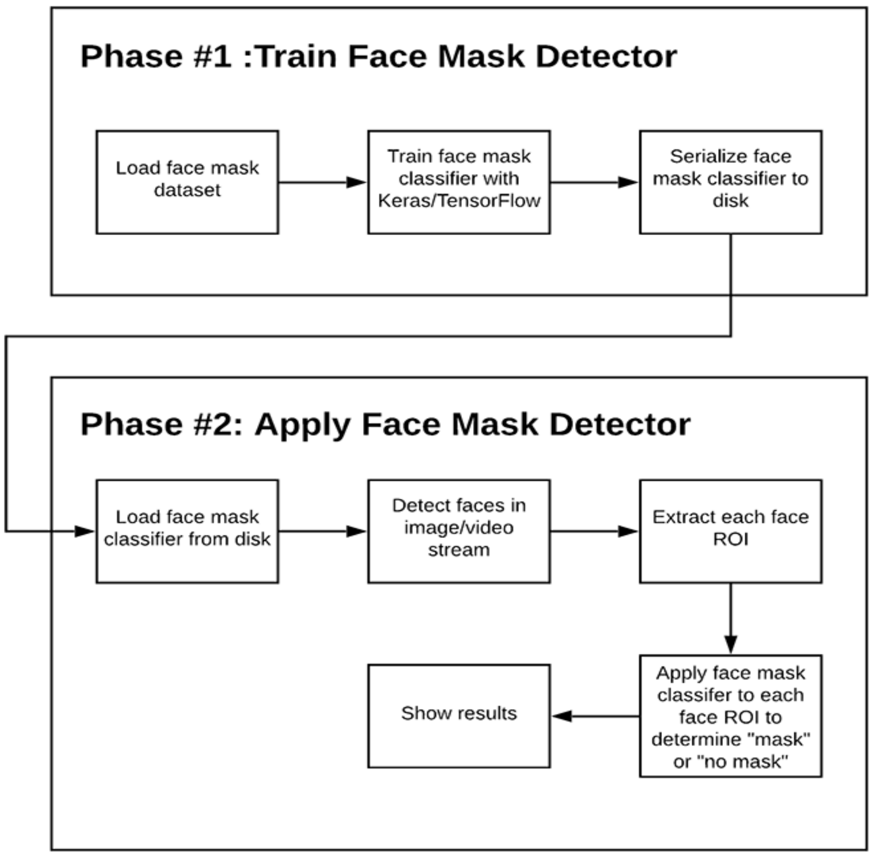

Phase #1 :Train Face Mask Detector
Load face mask dataset
Train face mask classifier with Keras/TensorFlow
Serialize face mask classifier to disk
Phase #2: Apply Face Mask Detector
Load face mask classifier from disk
Detect faces in image/video stream
Extract each face ROI
Apply face mask classifer to each face ROI to determine "mask" or "no mask"
Show results

Fig.1.1

In this project,there are to phases

1] Training 2]Deployment

1] Training:

Here we’ll focus on loading our face mask detection dataset from disk, training a model (using Keras/TensorFlow) 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.

2. REQUIREMENT ANALYSIS:

In this project first we need to get input image from user and then apply our face detector and mask detector models over them to predict whether the people detected in images are wearing mask or not. For this we are using already created python model for face detection and we have created mask detection model by ourself using mobilenet classification model.Mobilenet is very lightweight and hence can be run on devices with less graphical computation capacity at the cost of accuracy.

To create mask detecting models we are using dataset having images of people with mask and without mask. Normal images of people are used for with mask dataset and some of those images are processed using computer vision python script to apply mask on it so that we will get better accuracy in our prediction. The script creates landmarks on faces like eyes, mouth, nose and then using this landmarks mask are added in images.

CNN accuracy increases if we provide more dataset but because e have limited images we will be using data augmentation through which we will create multiple images buy scaling and rotating single image.

We are using customized fully connected layer which contains four sequential layers on top of the MobileNetV2 model was developed. The layers are:
1.Average Pooling layer with 7×7 weights
2.Linear layer with ReLu activation function
3.Dropout Layer
4.Linear layer with Softmax activation function with the result of 2 values.

The mid layer uses activation function relu to remove all negaaative values and final layer softmax function gives the result of two probabilities each one represents the classification of "mask" or "not mask".

3. PROJECT DESIGN

Flow Chart for face mask detection.

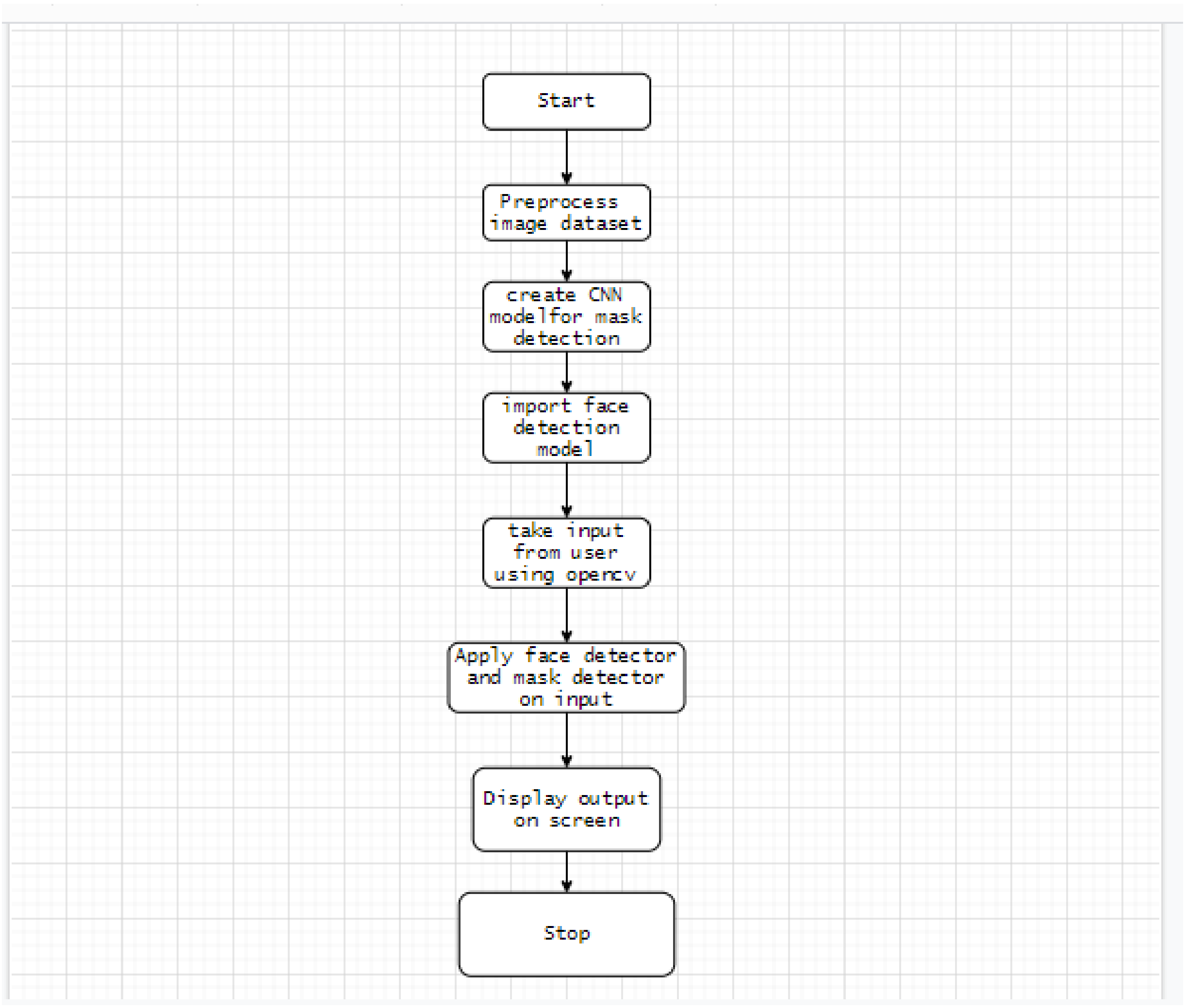

Start
Preprocess image dataset
create CNN modelfor mask detection
import face detection model
take input from user using opencv
Apply face detector and mask detector on input
Display output on screen
Stop

Fig 3.1

4.Technologies Used

4.1.Programming Language used :

Python and Jupyter LAB/Notebook.

4.2.Dataset used :

Dataset by BalagiShrinivisan from his github account.

Face-Mask-Detection/dataset at master · balajisrinivas/Face-Mask-Detection (github.com)

4.3.Model used :

We are using python predefined face detection model for the face detection.

https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20170830/res10_300x300_ssd_iter_140000.caffemodel

"https://github.com/opencv/opencv/blob/master/samples/dnn/face_detector/deploy.prototxt"

We are using mobilenet model to create our mask detector because it is lightweight and best suitable for machines with low graphical computational capacity.

We are using customized fully connected layer which contains four sequential layers on top of the MobileNetV2 model was developed.
The layers are:
1.Average Pooling layer with 7×7 weights
2.Linear layer with ReLu activation function
3.Dropout Layer
4.Linear layer with Softmax activation function with the result of 2 values.
The mid layer uses activation function relu to remove all negaaative values and final layer softmax function gives the result of two probabilities each one represents the classification of “mask” or “not mask”.

4.4. Packages used:

1.openCV

We are using opencv package for handling images and video, for resizing image as per requirement of mobilenet(224 x 224).

2.tensorflow.keras

we are using tensorflow.keras for creating our CNN using mobilenet binary classifier, to apply optimizer (dam) for our trained model and for preprocessing of input.

2.imutlis

We are using imutlis for directing our program throughout derectories for accessing images in our dataset.

3.sklearn

We are using sklearn for splitting dataset into training and testing dataset, preprocessing of data like converting labels into binary values and then to numpy arrays.

4.numpy

We are using numpy to convert our training dataset images and their labels to numpy arrays.

5.matplotlib

We are using matplotlib for displaying accuracy and loss of our trained model.

4.4.Data Visualization:

We are using matplot libraries for visualization of the data, to form charts to display the predicted price and also combining the predicted and actual price chart for comparison and opencv for displaying output image after applying face detector and mask detector on it.

If the video is provided as input then program will convert video into continuous image stream and display each image one by one after applying both detectors and if it is an image then rather than processing stream of images only single image is processed and displayed using opencv.imshow() function.

5.Results and Discussion

train_mask_detector

```
# 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"G:\study\pythonvscode\proj\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
print("[INFO] loading 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)

# 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")
```

detect_mask_image:

```python
# 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
from imutils.video import FileVideoStream
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"G:\study\pythonvscode\proj\face_detector\deploy.prototxt"
weightsPath = r"G:\study\pythonvscode\proj\face_detector\res10_300x300_ssd_ite
r_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("getting image...")
frame = cv2.imread("clg1.jpg")

for i in range(0,1):
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    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(10000)


# do a bit of cleanup
#cv2.destroyAllWindows()
#frame.stop()
```

Detect_mask:

```
# 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
from imutils.video import FileVideoStream
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"G:\study\pythonvscode\proj\face_detector\deploy.prototxt"
weightsPath = r"G:\study\pythonvscode\proj\face_detector\res10_300x300_ssd_ite
r_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
i=int(input("0-webcam 1-video"))
if(i == 0):
    print("[INFO] starting video stream...")
    vs = VideoStream(0).start()
elif(i == 1):
    vs=FileVideoStream("test1.mp4").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()
vs.stop()
```

Output:

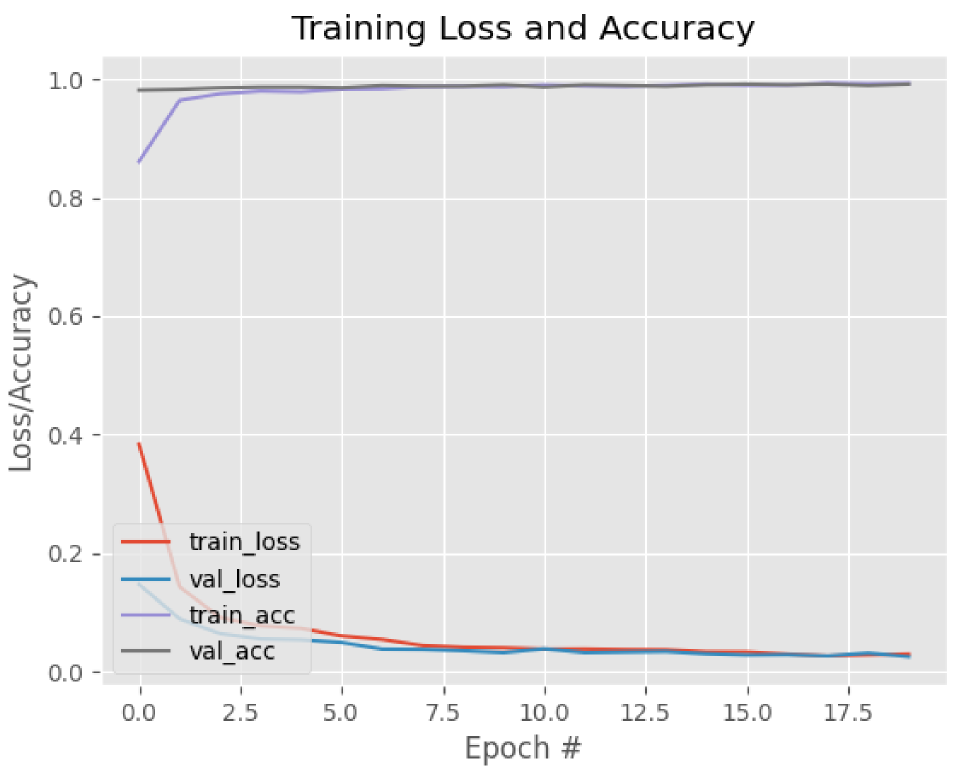


Fig 5.1

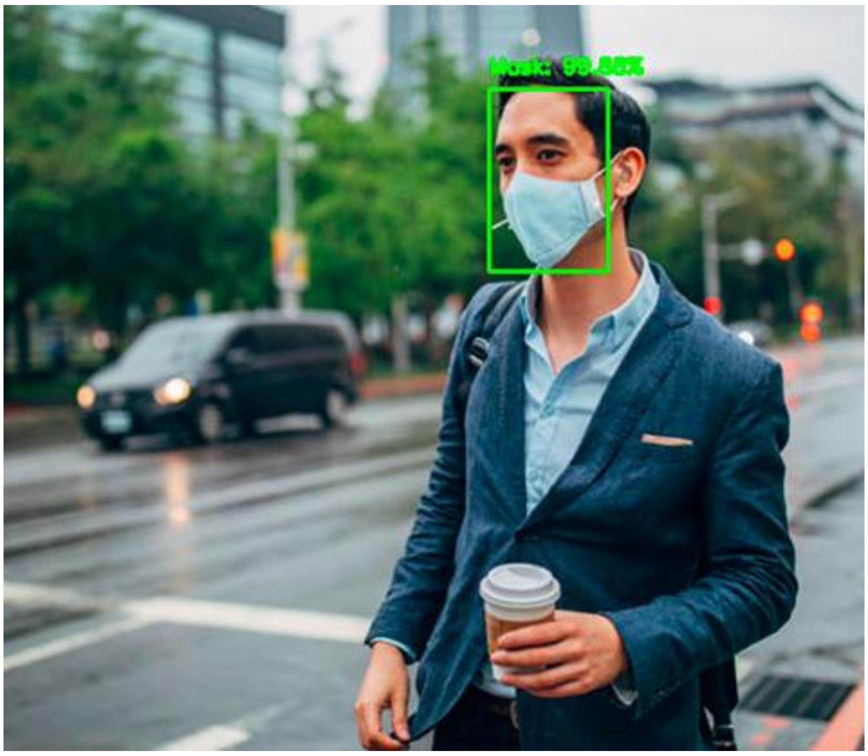
z 

Fig 5.2

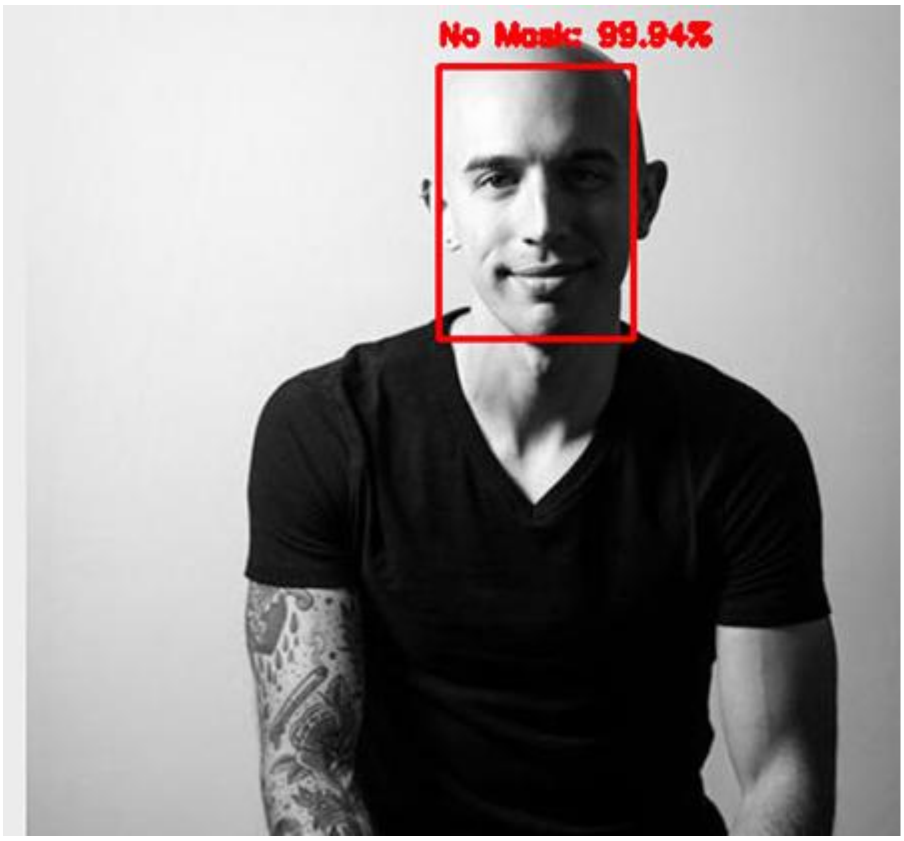


Fig 5.3

5.2.Discussion:

Here, we got Visualized charts of mask detection model. Here it can be seen we got almost 99% accuracy in our test cases and minimum loss. This model can be made more accurate by providing different types of training images because if the variety increases CNN will learn more about similar patterns in image.

6.Conclusion and Future scope -

6.1.Conclusion:

Finally we have created the face mask detection model by importing already created face detection model and creating mask detection model by ourselves. As we have seen in the result mobilenet classifier works just perfect in conjunction with CNN and the module got 99% accuracy but this accuracy is considered only for those images which are related to our training dataset, to improve that we have to add more different kind of images to our dataset.

In out sometimes we found that faces are not detected at all while processing its reason can be - the face is very small or cropped in original image or covered by something so that our imported model cannot detect that face. And because it is not detected program will not apply mask detector model on it and hence nothing will be displayed for that face in output. It can be improved by trying different face detector models.

6.2.Future Scope:

After the breakout of the worldwide pandemic covid-19, there arises a severe need of protection mechanisms , face mask being the primary one. The basic aim of this project is to detect the presence of mask on human faces over provided images so that it will be easy to enforce pandemic rule of compulsory masks in public places.

Right now the vaccination is developed for the covid-19 virus but it is not 100% cure so maybe this project can be used until the complete cure for this virus is created. And maybe after that also people will be aware about social distancing and this rules will be followed for further more years. So, there a good reason to improve functional capacity of this project and reduce as many errors in it as possible for its best use.

7.REFERENCES

1.**Adrian Rosebrock** pyimagesearch> covid-19 face mask detector on May 4, 2020

COVID-19: Face Mask Detector with OpenCV, Keras/TensorFlow, and Deep Learning - PyImageSearch

2.Ramachandran K · Ideas2IT>Face mask detector using deep learning on May 28, 2020

Face Mask Detector using Deep Learning (PyTorch) and Computer Vision (OpenCV) - Ideas2IT

3.balajishrinivasanBalajiSrinivasan>face mask detection on realtime video stream

(280) Face Mask Detection using Python, Keras, OpenCV and MobileNet | Detect masks real-time video streams - YouTube