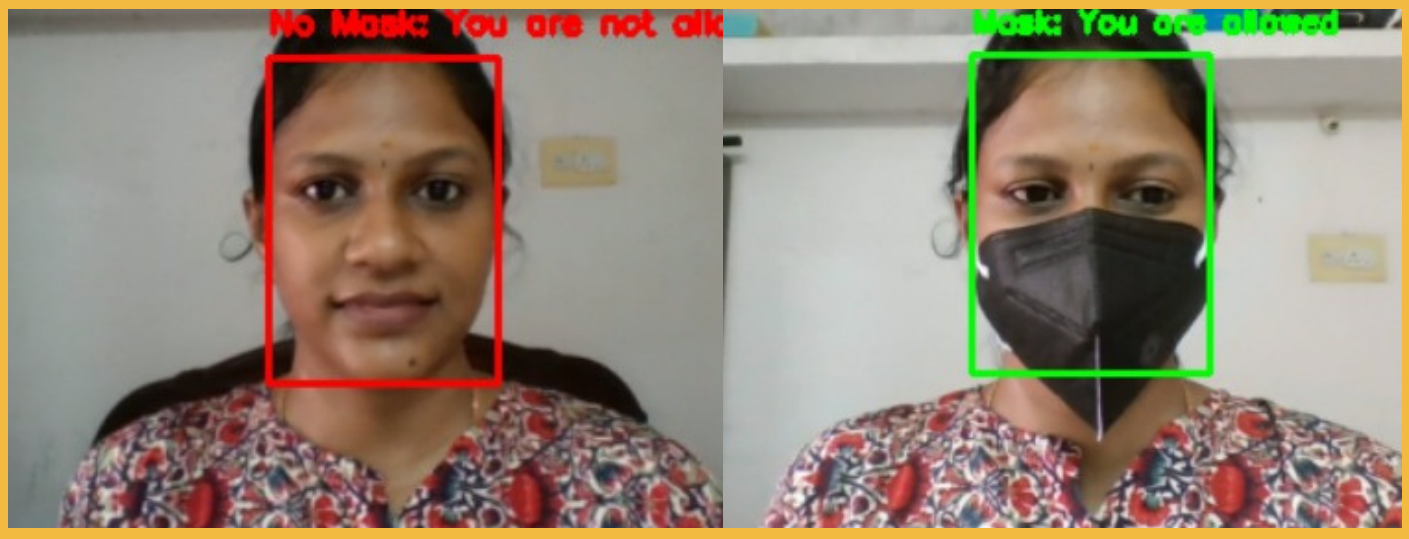
Face Mask Detection Using Tensorflow



Slide for the steps

STEPS TO ATTAIN ABOVE RESULT ..!

STEP 1 : Data Collection & Preprocessing 📊

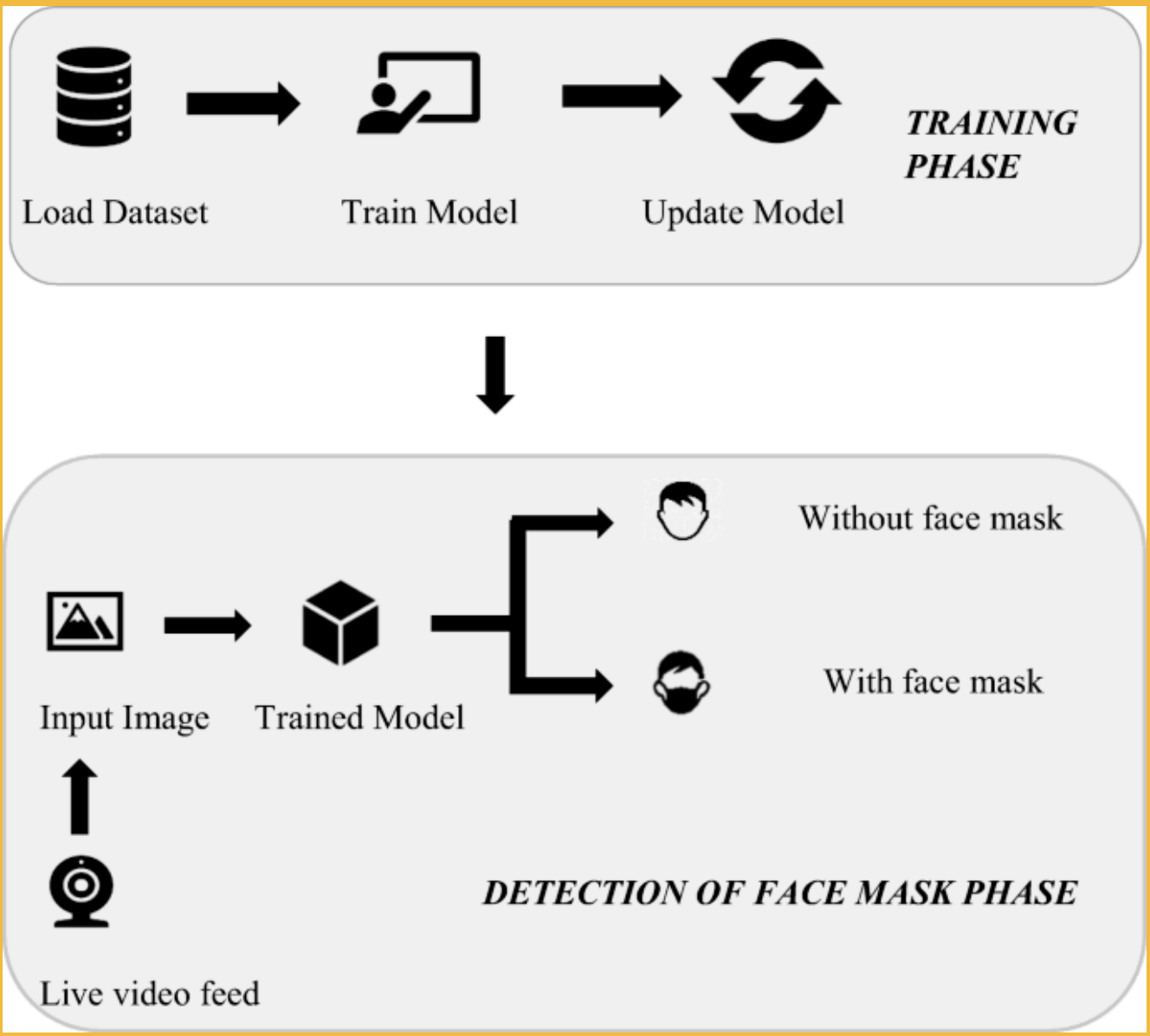
i.Dataset Selection
ii.Data Augmentation
iii.Image Preprocessing

- Used a dataset that includes labeled images of individuals both with and without masks.
- The dataset was balanced between the two classes: "Mask" and "No Mask."

STEP 2 : Model Creation & Training

i.Model Architecture

ii:Training the Model



STEP 3: Face Detection Integration with OpenCV

i.Real-Time Face Detection
ii.Extracting Face ROI

STEP 4:Mask Detection & Bounding Box

i.Mask Prediction

ii:Drawing the Bounding Box & Label

iii.Real time feedback

STEP 5: Deployment & Optimization

i.System Deployment
ii.Performance Optimization

Code for train the model

In [33]:
```
# 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_inpu
t 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 argparse import os
```

In [34]:
```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 10
BS = 32
```

In [36]:
```
# r before string will prevent Python from interpreting escape characters.
DIRECTORY = r"D:\Deep Learning\facemask\dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

In [37]:
```
# 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 = []
```

```
[INFO] loading images...
```

In [38]:
```
from imutils import paths

# Define the path to your dataset
dataset_path = r"D:\Deep Learning\facemask\dataset"
# List all image paths in the dataset
imagePaths = list(paths.list_images(dataset_path))
# Print the number of images found and a few example paths
print(f"Found {len(imagePaths)} images.")
print("Example image paths:")
print(imagePaths[:5])  # Print first 5 image paths
```

```
Found 3846 images.
Example image paths:
['D:\\Deep Learning\\facemask\\dataset\\without_mask\\0.jpg', 'D:\\Deep Learning\\facemask\\dataset\\without_mas
k\\0_0_aidai_0014.jpg', 'D:\\Deep Learning\\facemask\\dataset\\without_mask\\0_0_aidai_0029.jpg', 'D:\\Deep Lear
ning\\facemask\\dataset\\without_mask\\0_0_aidai_0043.jpg', 'D:\\Deep Learning\\facemask\\dataset\\without_mask\
\0_0_aidai_0074.jpg']
```

In [39]:
```
# loop over the image paths
for imagePath in imagePaths:
        # extract the class label from the filename
        label = imagePath.split(os.path.sep)[-2]
        # load the input image (224x224) and preprocess it
        image = load_img(imagePath, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)
        # update the data and labels lists, respectively
        data.append(image)
        labels.append(label)



# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
```

```
C:\anaconda\envs\myenv\lib\site-packages\PIL\Image.py:992: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

In [40]:
```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
```

In [41]:
```
# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)
```

In [43]:
```
# 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")
```

In [44]:
```
# Construct the validation image generator
val_aug = ImageDataGenerator()
```

In [45]:
```
# Create data generators
train_generator = aug.flow(trainX, trainY, batch_size=BS)
val_generator = val_aug.flow(testX, testY, batch_size=BS)
```

In [46]:
```
# 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)))
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 [==============================] - 1s 0us/step
```

In [50]:
```
# 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)
```

In [51]:
```
# 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)
```

In [52]:
```
# 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
```

In [53]:
```
# compile our model
print("[INFO] compiling model...")
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
[INFO] compiling model...
```

In [54]:
```
# Compile our model
print("[INFO] compiling model...")
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
[INFO] compiling model...
```

In [56]:
```
# Train the head of the network
print("[INFO] training head...")
H = model.fit(
    train_generator,
    steps_per_epoch=len(trainX) // BS,
    validation_data=val_generator,
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

```
[INFO] training head...
Epoch 1/10
96/96 [==============================] - 60s 627ms/step - loss: 0.0441 - accuracy: 0.9872 - val_loss: 0.0406 - v
al_accuracy: 0.9909
Epoch 2/10
96/96 [==============================] - 60s 630ms/step - loss: 0.0415 - accuracy: 0.9875 - val_loss: 0.0380 - v
al_accuracy: 0.9883
Epoch 3/10
96/96 [==============================] - 59s 618ms/step - loss: 0.0405 - accuracy: 0.9875 - val_loss: 0.0363 - v
al_accuracy: 0.9896
Epoch 4/10
96/96 [==============================] - 60s 621ms/step - loss: 0.0408 - accuracy: 0.9872 - val_loss: 0.0344 - v
al_accuracy: 0.9896
Epoch 5/10
96/96 [==============================] - 60s 629ms/step - loss: 0.0340 - accuracy: 0.9882 - val_loss: 0.0346 - v
al_accuracy: 0.9909
Epoch 6/10
96/96 [==============================] - 61s 638ms/step - loss: 0.0292 - accuracy: 0.9918 - val_loss: 0.0364 - v
al_accuracy: 0.9896
Epoch 7/10
96/96 [==============================] - 69s 717ms/step - loss: 0.0312 - accuracy: 0.9911 - val_loss: 0.0336 - v
al_accuracy: 0.9883
Epoch 8/10
96/96 [==============================] - 62s 645ms/step - loss: 0.0327 - accuracy: 0.9901 - val_loss: 0.0334 - v
al_accuracy: 0.9883
Epoch 9/10
96/96 [==============================] - 60s 621ms/step - loss: 0.0323 - accuracy: 0.9882 - val_loss: 0.0325 - v
al_accuracy: 0.9909
Epoch 10/10
96/96 [==============================] - 60s 627ms/step - loss: 0.0270 - accuracy: 0.9921 - val_loss: 0.0351 - v
al_accuracy: 0.9935
```

In [57]:
```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
```

```
[INFO] evaluating network...
25/25 [==============================] - 11s 421ms/step
```

In [58]:
```
# 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)
```

In [59]:
```
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
        target_names=lb.classes_))
```

```
              precision    recall  f1-score   support

   with_mask       0.99      1.00      0.99       384
without_mask       1.00      0.99      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
```

In [60]:
```
# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("Face_Mask_Detector.h5")
```

```
[INFO] saving mask detector model...
```

In [65]:
```
# 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")
```

In [ ]:

In [ ]:

Code for test the model

In [ ]:
```
# 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
```

In [ ]:
```
#import cv2
#import numpy as np
# Create a black image
#image = np.zeros((300, 300, 3), dtype=np.uint8)
# Add a white rectangle
#cv2.rectangle(image, (50, 50), (250, 250), (255, 255, 255), -1)
# Display the image
#cv2.imshow('Test Window', image)
#cv2.waitKey(0)
#cv2.destroyAllWindows()
```

In [ ]:
```
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    (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()

    # 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 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
        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)
```

In [ ]:
```
# load our serialized face detector model from disk
prototxtPath = r"D:\Deep Learning\facemask\face_detector\deploy.prototxt"
weightsPath = r"D:\Deep Learning\facemask\face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

In [ ]:
```
from tensorflow.keras.models import load_model
import os
# Path to the mask detector model
model_path = r"D:\Deep Learning\facemask\Face_Mask_Detector.h5"
# Check if the model file exists
if os.path.exists(model_path):
    # Load the face mask detector model from disk
    maskNet = load_model(model_path)
    print("Model loaded successfully.")
else:
    print(f"Error: The model file does not exist at path {model_path}")
```

In [ ]:
```
# load the face mask detector model from disk
maskNet = load_model(r"D:\Deep Learning\facemask\Face_Mask_Detector.h5")
```

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

In [ ]:
```
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    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)

        # display the label and bounding box rectangle on the output frame
        if(label == "Mask"):
            cv2.putText(frame, "Mask: You are allowed", (startX, startY - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
        elif(label == "No Mask"):
            lab = "No Mask: You are not allowed"
            cv2.putText(frame, lab, (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()
```

In [ ]:

In [ ]:

In [ ]:

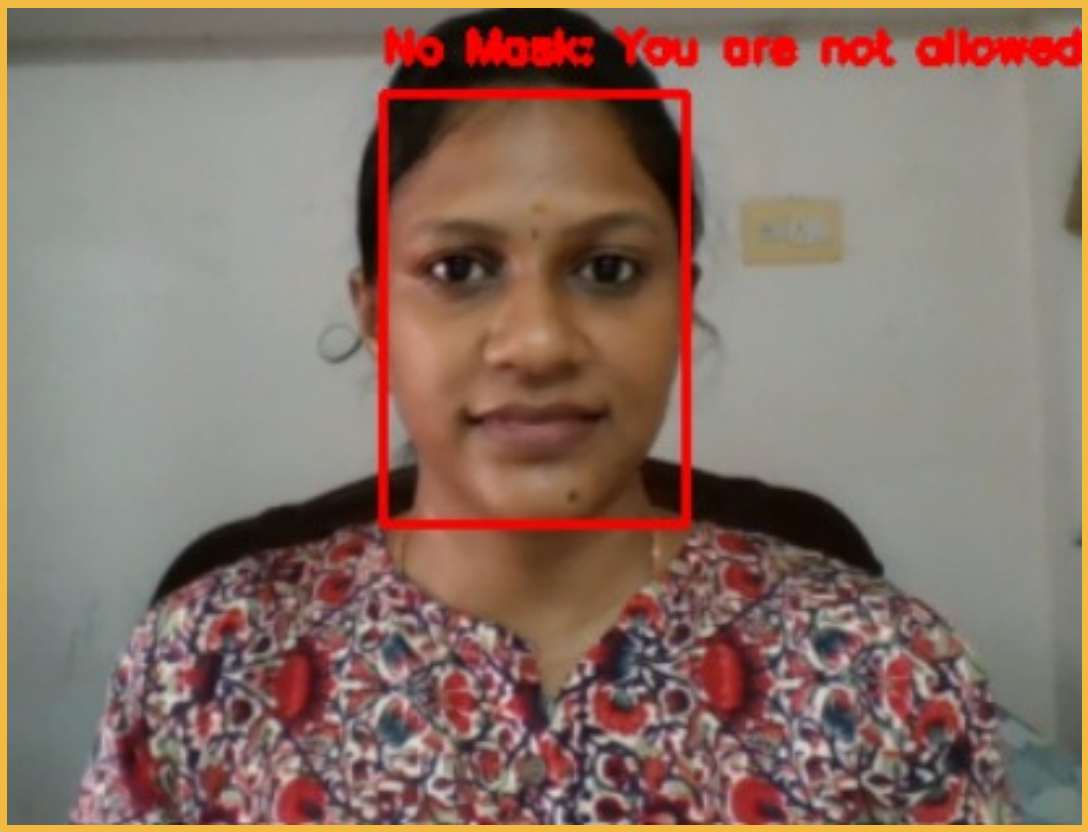
In [ ]:

In [ ]:

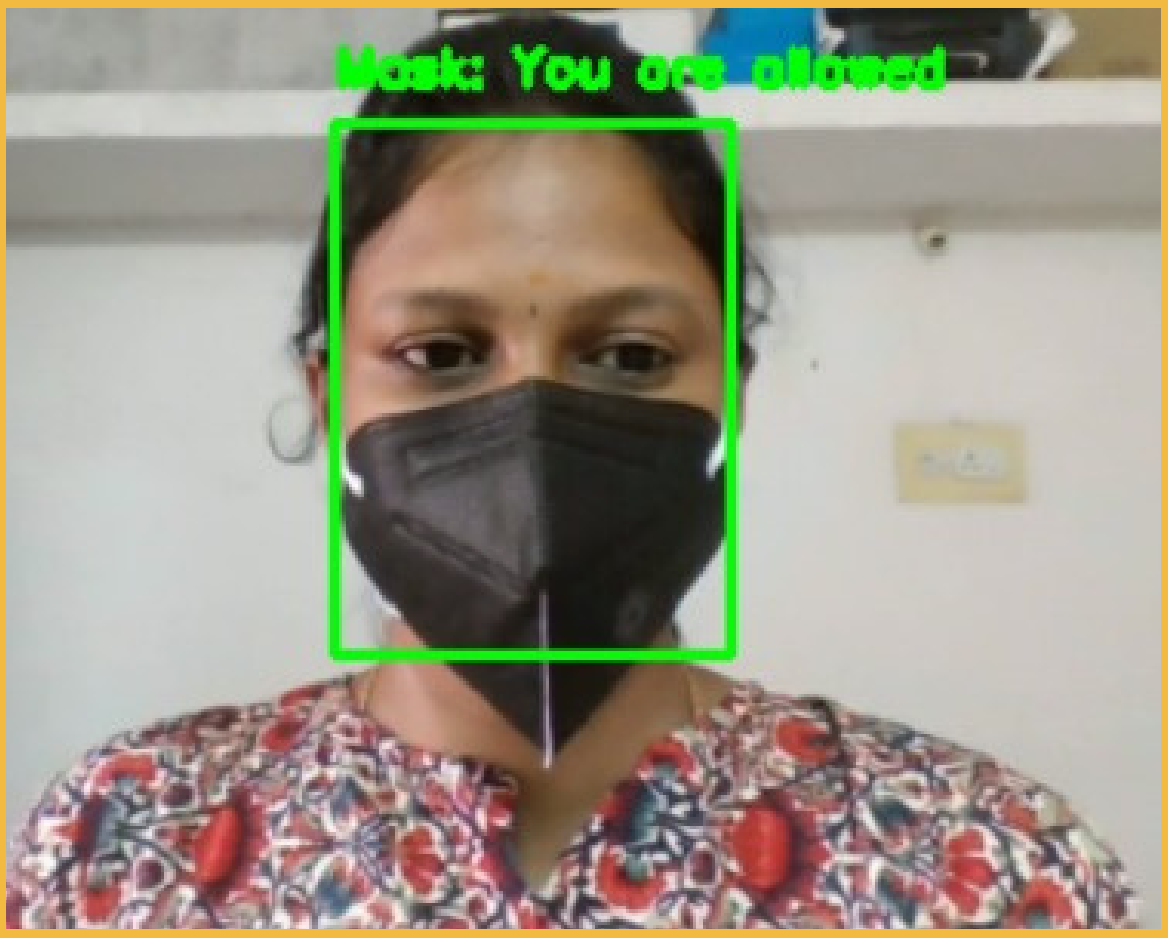
In [ ]:

Output of this Face Mask Detection project is

No Mask : You are not allowed



Mask : You are allowed



If you like this post do follow me on LinkedIn 

https://www.linkedin.com/in/umadevi-vivekananthan