A Project/Dissertation ETE Report

on

Mask Recognition Technology Using AI/ML

Submitted in partial fulfillment of the
requirement for the award of the degree
of

B.Tech. (CSE)

GALGOTIAS
UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision
of the Name of
Supervisor Dr. Vipin
Rai
Designation:
Associate Professor

Submitted By
Ayush Bhatt
18SCSE1010177
Ankur Bhatnagar
18SCSE1010240

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
December,2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled " **Mask Recognition Technology Using AI/ML** " in partial fulfillment of the requirements for the award of the BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of JULY-2021 to DECEMBER-2021, under the supervision of Dr.Vipin Rai, Associate Professor, Department of Computer Science and Engineering of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

18SCSE1010177 - AYUSH BHATT

18SCSE1010240 - ANKUR BHATNAGAR

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor
(Dr.Vipin Rai, Associate Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **18SCSE1010177 - Ayush Bhatt ,18SCSE1010240- Ankur Bhatnagar** has been held on _ ______and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Place:

Abstract

COVID-19 pandemic has rapidly affected our day-to-day life disrupting world trade and movements. Wearing a protective face mask has become a new normal. In the near future, many public service providers will ask the customers to wear masks correctly to avail of their services. Therefore, face mask detection has become a crucial task to help global society. This paper presents a simplified approach to achieve this purpose using some basic Machine Learning packages like TensorFlow, Keras, OpenCV . The proposed method detects the face from the image correctly and then identifies if it has a mask on it or not. As a surveillance task performer, it can also detect a face along with a mask in motion. The method attains accuracy up to 95.77% and 94.58% respectively on two different datasets. We explore optimized values of parameters using the Sequential Convolutional Neural Network model to detect the presence of masks correctly without causing over-fitting.

In this work, a deep learning-based approach for detecting masks over faces in public places to curtail the community spread of Coronavirus is presented. The proposed technique efficiently handles occlusions in dense situations by making use of an ensemble of single and two-stage detectors at the pre-processing level. The ensemble approach not only helps in achieving high accuracy but also improves detection speed considerably. Furthermore, the application of transfer learning on pre-trained models with extensive experimentation over an unbiased dataset resulted in a highly robust and low-cost system. The identity detection of faces, violating the mask norms further, increases the utility of the system for public benefits.

Finally, the work opens interesting future directions for researchers. Firstly, the proposed technique can be integrated into any high-resolution video surveillance devices and not limited to mask detection only. Secondly, the model can be extended to detect facial landmarks with a facemask for biometric purposes.

Table of Content

List Of Figures

Chapter-1 Introduction

According to the World Health Organization (WHO) official Status Report - 205, coronavirus 2019 (COVID-19) has infected more than 20 million people worldwide causing more than 0.7 million deaths. People with COVID-19 had a number of reported symptoms - ranging from mild to severe illness. Respiratory problems such as shortness of breath or difficulty breathing are one of them. Adults with pneumonia may be more prone to COVID-19 complications as they appear to be at higher risk. Other common human coronaviruses that infect humans worldwide are 229E, HKU1, OC43, and NL63. Prior to the demise of humans, viruses such as the 2019-nCoV, SARS-CoV, and MERS-CoV infect animals and turn them into human coronaviruses. People with respiratory problems can expose anyone (close to them) to a contagious bead. Circumcision of an infected person can cause transmission of communication as droplets carrying the virus may reach its immediate vicinity.

To prevent certain respiratory infections, including COVID-19, wearing a clinical mask is very necessary. The public should be aware that you must wear a mask to control the source or dislike COVID-19. Points that may be of interest to the use of masks are to reduce the risk of injury from a dangerous person during the "pre-symptomatic period" and to discriminate against sophisticated people who wear a mask to prevent the spread of the virus. The WHO emphasizes the prioritization of medical masks and respirators for health care Associates. Therefore, the discovery of a face mask has become an important activity in today's world society.

Finding a face mask involves finding the location of the face and finding out if it has a mask on it or not. The problem is with the acquisition of a common object to determine the categories of objects. Phase identification is associated with the division of a particular business group namely Face. It has many applications, such as automatic driving, education, surveillance, and more . This paper introduces a simplified way to achieve the above goal using basic Machine Learning (ML) packages such as TensorFlow, Keras, OpenCV and Scikit-Learn.

Chapter-2 Literature Survey

1: MAMATA S. KALAS, REAL TIME FACE DETECTION AND TRACKING USING OPENCV, International Journal of Soft Computing and Artificial Intelligence, ISSN: 2321-404X, Volume-2, Issue-1, May- 2014

Haar- like feature: Haar-like wavelets are binary rectangular representations of 2D waves. A common visual representation is by black (for value minus one) and white (for value plus one) rectangles. The square above the 0-1- interval shows the corresponding Haar-like wavelet in common black-white representation. The rectangular masks used for visual object detection are rectangles tessellated by black and white smaller rectangles. Those masks are designed in correlation to visual recognition tasks to be solved, and known as a Haar-like feature each call, a distribution of weights is updated that indicates the importance of examples in the data set for the classification. On each round, the weights of each incorrectly classified example are increased, and the weights of each correctly classified example are decreased, so the new classifier focuses on the examples which have so far eluded correct classification.

2:Walid Hariri, Efficient Masked Face Recognition Method during the COVID-19 Pandemic, pp.1-7, July 2020

Pre-processing and cropping filter: The images of the dataset are already cropped around the face, so there is no need for a face detection stage to localize the face from each image. To do so, we detect 68 facial landmarks using Dlib-ml open- source library. According to the eyes location, we apply a 2D rotation to make them horizontal. The next step is to apply a cropping filter in order to extract only the non- masked region. To do so, we firstly normalize all face images into 240 x 240 pixels. Next, we use the partition into blocks. The principle of this technique is to divide the image into 100 fixed-size square blocks (24 x 24 pixels in our case). Then we extract only the blocks including the non-masked region (blocks from number 1 to 50). Finally, we eliminate the rest of the numbers of the blocks.

3:Vinitha.V1, Valentina.V2, COVID-19 FACEMASK DETECTION WITH DEEP LEARNING AND COMPUTER VISION, International Research Journal of Engineering and Technology (IRJET), Volume: 07, pp.1-6, Aug 2020.

The proposed system focuses on how to identify the person on an image/video stream wearing a face mask with the help of computer vision and deep learning algorithms by using the OpenCV, Tensor flow, Keras and PyTorch libraries. The majority of the images were augmented by OpenCV. The set of images were already labeled mask and no mask.

The images that were present were of different sizes and resolutions, probably extracted from different sources or from machines (cameras) of different resolutions.

4: S. Ghosh, N. Das and M. Nasipuri, "Reshaping inputs for convolutional neural network: Some common and uncommon methods", *Pattern Recognition*, vol. 93, pp. 79-94, 2019.

convolutional neural network (CNNs) in computer vision comes with a strict constraint regarding the size of the input image. The prevalent practice reconfigures the images before fitting them into the network to surmount the inhibition.

Here the main challenge of the task is to detect the face from the image correctly and then identify if it has a mask on it or not. In order to perform surveillance tasks, the proposed method should also detect a face along with a mask in motion.

Tools Used

A. TensorFlow

TensorFlow, an interface for expressing machine learning algorithms, is utilized for implementing ML systems into fabrication over a bunch of areas of computer science, including sentiment analysis, voice recognition, geographic information extraction, computer vision, text summarization, information retrieval, computational drug discovery and flaw detection to pursue research . In the proposed model, the whole Sequential CNN architecture (consists of several layers) uses TensorFlow at backend. It is also used to reshape the data (image) in the data processing.

B. Keras

Keras gives fundamental reflections and building units for creation and transportation of ML arrangements with high iteration velocity. It takes full advantage of the scalability and cross-platform capabilities of TensorFlow. The core data structures of Keras are layers and models . All the layers used in the CNN model are implemented using Keras. Along with the conversion of the class vector to the binary class matrix in data processing, it helps to compile the overall model.

C. OpenCV

OpenCV (Open Source Computer Vision Library), an open-source computer vision and ML software library, is utilized to differentiate and recognize faces, recognize objects, group movements in recordings, trace progressive modules, follow eye gesture, track camera actions, expel red eyes from pictures taken utilizing flash, find comparative pictures from an image database, perceive landscape and set up markers to overlay it with increased reality and so forth . The proposed method makes use of these features of OpenCV in resizing and color conversion of data images.

DataSet

Two data sets were used to check the current method. Dataset 1 has 1376 photos with 690 photos of people wearing face masks and another 686 photos of people without wearing face masks. Figure 1 usually consists of the shape of the front face and one face without a mask.

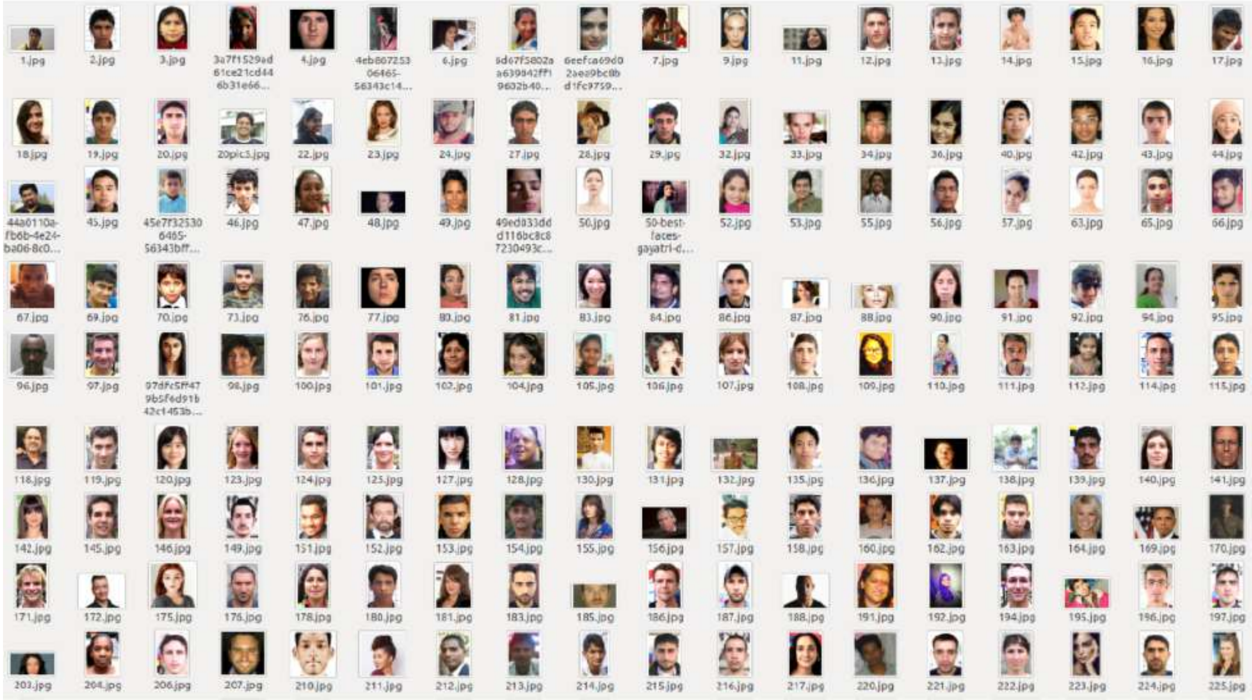


Fig 1

Dataset 2 from Kaggle contains 853 images and its contents are highlighted with or without a mask. On the figure 2 Other face masks are head turns, tilt and tilt with multiple faces on the frame and different types of masks with different colors as well.

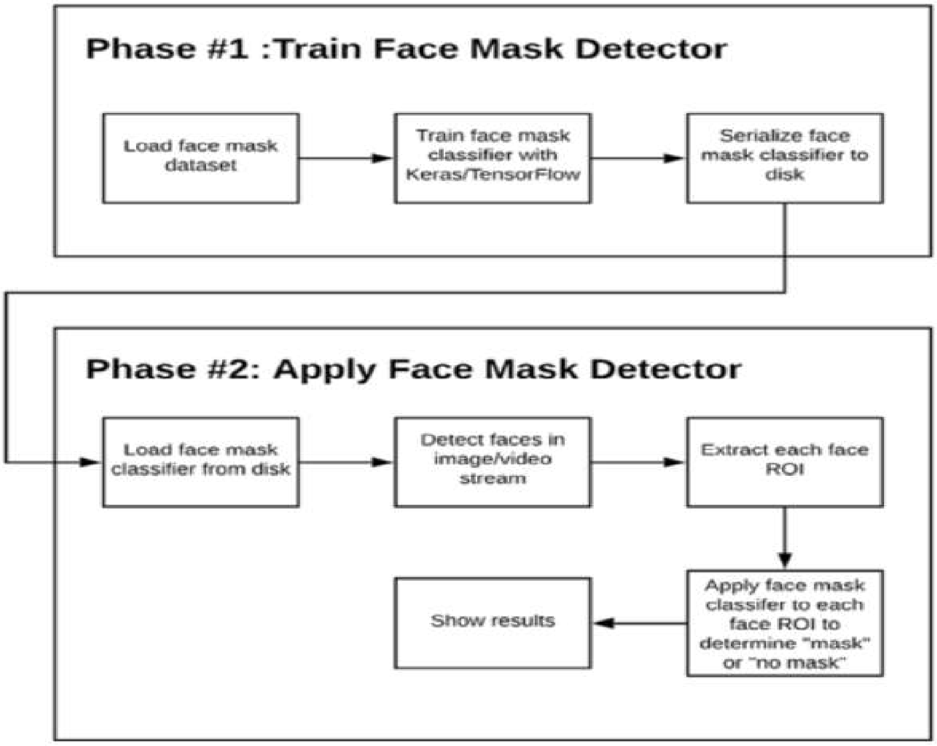
-110603108-gettyima...
-110695084-mask2.jpg
0-with-mask.jpg
1Nl-1-surgical-mask-whi...
1-respirator-mask.jpg
2ca8e9fa7b11fae2a0b15e83c318...
2-with-mask.jpg
3-ply-tie-on-surgical-mask-yell...
3-with-mask.jpg
4-with-mask.jpg
6-with-mask.jpg
7-with-mask.jpg
8-with-mask.jpg
9-with-mask.jpg
10Pcs-pack-Disposable-Non-Wov...
10-with-mask.jpg
12-with-mask.jpg
13-with-mask.jpg
14-with-mask.jpg
15-with-mask.jpg
16-with-mask.jpg
17-with-mask.jpg
18-with-mask.jpg
19-with-mask.jpg
21-with-mask.jpg
24-with-mask.jpg
25a0bb7104f6291c2737cdbe3e...
27-with-mask.jpg
29-with-mask.jpg
31-with-mask.jpg
32-with-mask.jpg
33-with-mask.jpg
35-with-mask.jpg
37-with-mask.jpg
38-with-mask.jpg
39a923aa.jpg
39-with-mask.jpg
40-with-mask.jpg
41-with-mask.jpg
41yveM7kuVL._AC_.jpg
42-with-mask.jpg
43-with-mask.jpg
45-with-mask.jpg
46-with-mask.jpg
47-with-mask.jpg
50-with-mask.jpg
51pLp2uev9L.jpg
52-with-mask.jpg
53-with-mask.jpg
56-with-mask.jpg
59-with-mask.jpg
61-with-mask.jpg
61XJbVDexKL._AC_SL1500_.jpg
64-with-mask.jpg
65-with-mask.jpg
67-with-mask.jpg
68-with-mask.jpg
70-with-mask.jpg
71CETHshWPL._AC_SY679_.jpg
71-with-mask.jpg
72-with-mask.jpg
74-with-mask.jpg
75-with-mask.jpg
76-with-mask.jpg
77-with-mask.jpg
79-with-mask.jpg
80-with-mask.jpg
81-with-mask.jpg
82-with-mask.jpg
83-with-mask.jpg
86-with-mask.jpg
88-with-mask.jpg
89-with-mask.jpg
92-with-mask.jpg
93-with-mask.jpg
94-with-mask.jpg
96-with-mask.jpg
97-with-mask.jpg
99-with-mask.jpg
100-with-mask.jpg
101-with-mask.jpg
103-with-mask.jpg
104-with-mask.jpg
105-with-mask.jpg
106.jpg
107-with-mask.jpg
108-with-mask.jpg
109-with-mask.jpg
112-with-mask.jpg
113-with-mask.jpg
115-with-mask.jpg
116-with-mask.jpg
117-with-mask.jpg
118-with-mask.jpg
119-with-mask.jpg
120-with-mask.jpg
122-with-mask.jpg
123-with-mask.jpg
124-with-mask.jpg
125-with-mask.jpg
126-with-mask.jpg
128-with-mask.jpg
129-with-mask.jpg
131-with-mask.jpg
132-with-mask.jpg
133-with-mask.jpg
134-with-mask.jpg
135-with-mask.jpg
136-with-mask.jpg
137-with-mask.jpg
138-with-mask.jpg
139-with-mask.jpg
140-with-mask.jpg
143-with-mask.jpg
144-with-mask.jpg
147-with-mask.jpg
148-with-mask.jpg
150-with-mask.jpg
151-with-mask.jpg
152-with-mask.jpg
153-with-mask.jpg
154-with-mask.jpg
155-with-mask.jpg
156-with-mask.jpg
158-with-mask.jpg
159-with-mask.jpg
161-with-mask.jpg
162-with-mask.jpg
163-with-mask.jpg
168-with-mask.jpg
169-with-mask.jpg
170-with-mask.jpg
171-with-mask.jpg
172-with-mask.jpg
173-with-mask.jpg
175-with-mask.jpg
176-with-mask.jpg
177-with-mask.jpg
179-with-mask.jpg
180-with-mask.jpg
182-with-mask.jpg
183-with-mask.jpg
185-with-mask.jpg
187-with-mask.jpg

Fig 2

Project Methodology

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 (as shown by Figure 1 below):

- 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
- 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


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

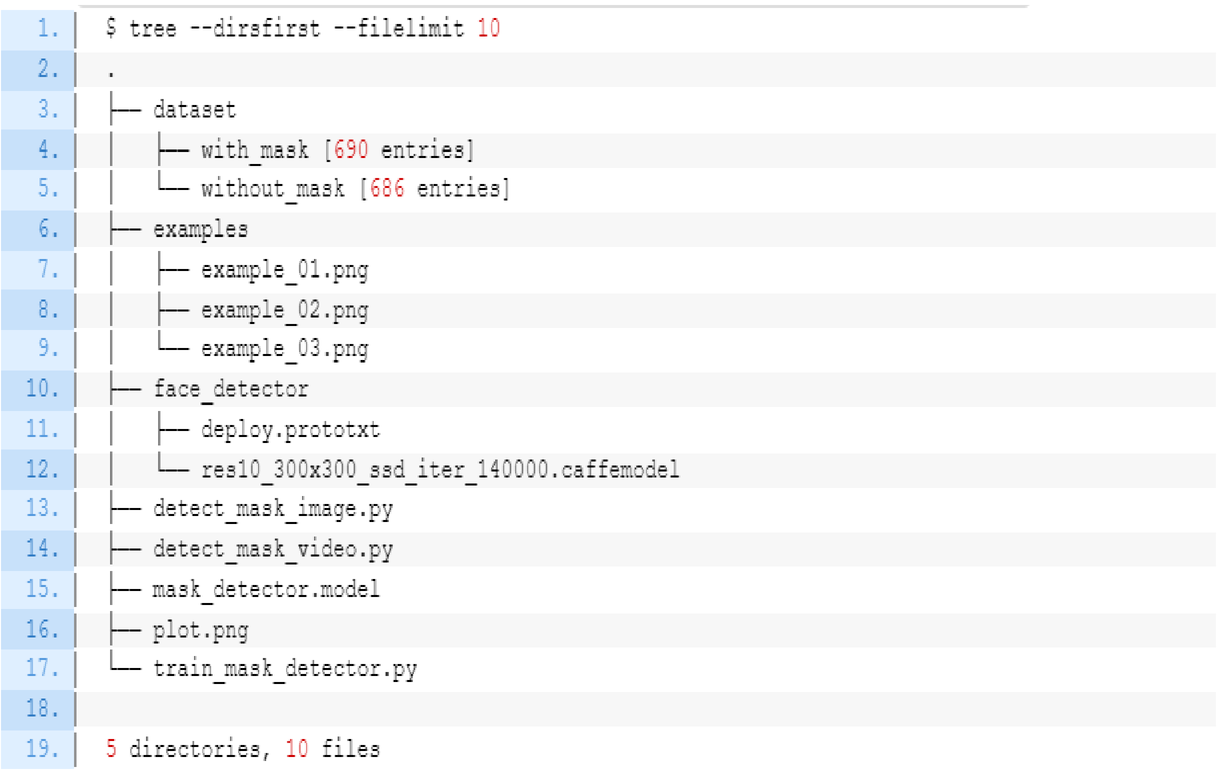
We’ll review each of these phases and associated subsets in detail in the remainder of this tutorial, but in the meantime, let’s take a look at the dataset we’ll be using to train our COVID-19 face mask recogniser.

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

- with_mask: 690 images
- without_mask: 686 images

Our goal is to train a custom deep learning model to detect whether a person is or is not wearing a mask.

Project Structure



```
$ tree --dirsfirst --filelimit 10
.
├── dataset
│   ├── with_mask [690 entries]
│   └── without_mask [686 entries]
├── examples
│   ├── example_01.png
│   ├── example_02.png
│   └── example_03.png
├── face_detector
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
├── detect_mask_image.py
├── detect_mask_video.py
├── mask_detector.model
├── plot.png
└── train_mask_detector.py

5 directories, 10 files
```

The dataset/ directory contains the data described in the “Our COVID-19 face mask detection dataset” section.

Three image examples/ are provided so that you can test the static image face mask detector.

We’ll be reviewing three Python scripts in this:

train_mask_detector.py: Accepts our input dataset and fine-tunes MobileNetV2 upon it to create our mask_detector.model. A training history plot.png containing accuracy/loss curves is also produced detect_mask_image.py: Performs face mask detection in static images detect_mask_video.py: Using your webcam, this script applies face mask detection to every frame in the stream

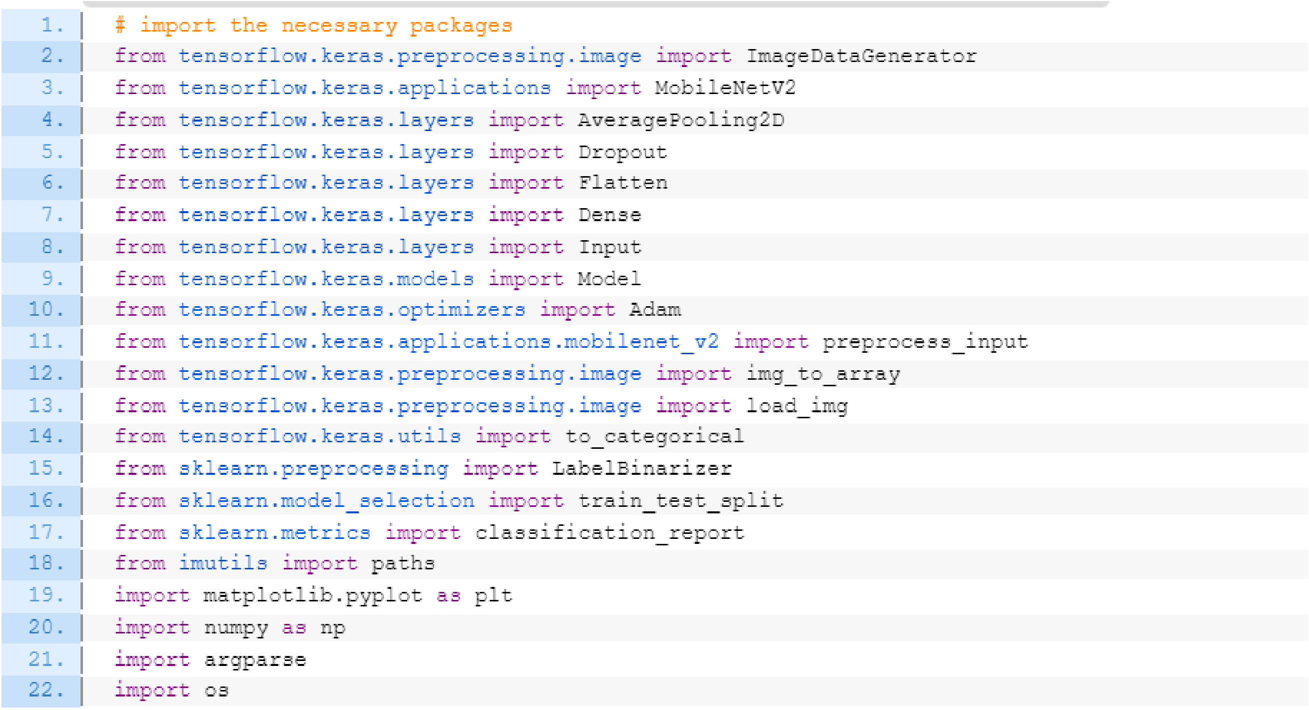
In the next two sections, we will train our face mask detector.

Implementing our COVID-19 face mask detector training script with Keras and TensorFlow

Now that we've reviewed our face mask dataset, let's learn how we can use Keras and TensorFlow to train a classifier to automatically detect whether a person is wearing a mask or not.

To accomplish this task, we'll be fine-tuning the MobileNet V2 architecture, a highly efficient architecture that can be applied to embedded devices with limited computational capacity (ex., Raspberry Pi, Google Coral, NVIDIA Jetson Nano, etc.).

Deploying our face mask detector to embedded devices could reduce the cost of manufacturing such face mask detection systems, hence why we choose to use this architecture.



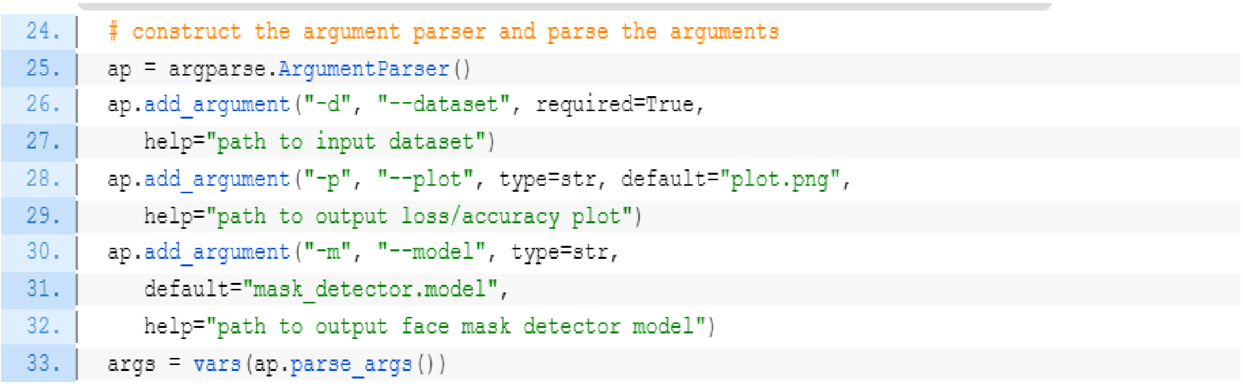
```python
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
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 argparse
import os
```

Our set of tensorflow.keras imports allow for:

- Data augmentation
- Loading the MobilNetV2 classifier (we will fine-tune this model with pre-trained ImageNet weights)
- Building a new fully-connected (FC) head
- Pre-processing
- Loading image data

We'll use scikit-learn (sklearn) for binarizing class labels, segmenting our dataset, and printing a classification report.

My imutils paths implementation will help us to find and list images in our dataset. And we'll use matplotlib to plot our training curves.



```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to output face mask detector model")
args = vars(ap.parse_args())
```

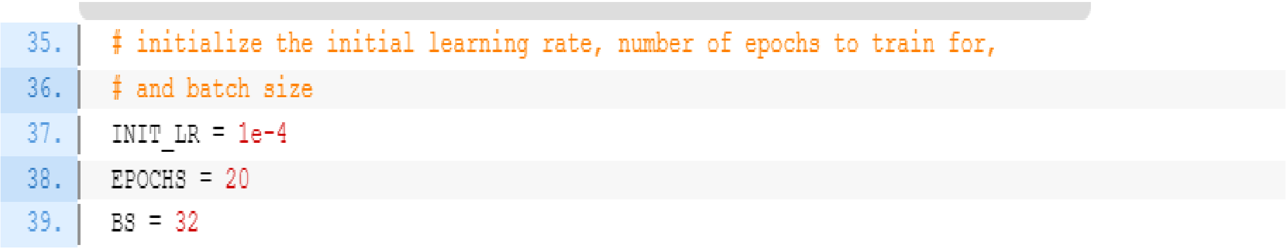
Our command line arguments include:

--dataset: The path to the input dataset of faces and and faces with masks

--plot: The path to your output training history plot, which will be generated using matplotlib

--model: The path to the resulting serialized face mask classification model

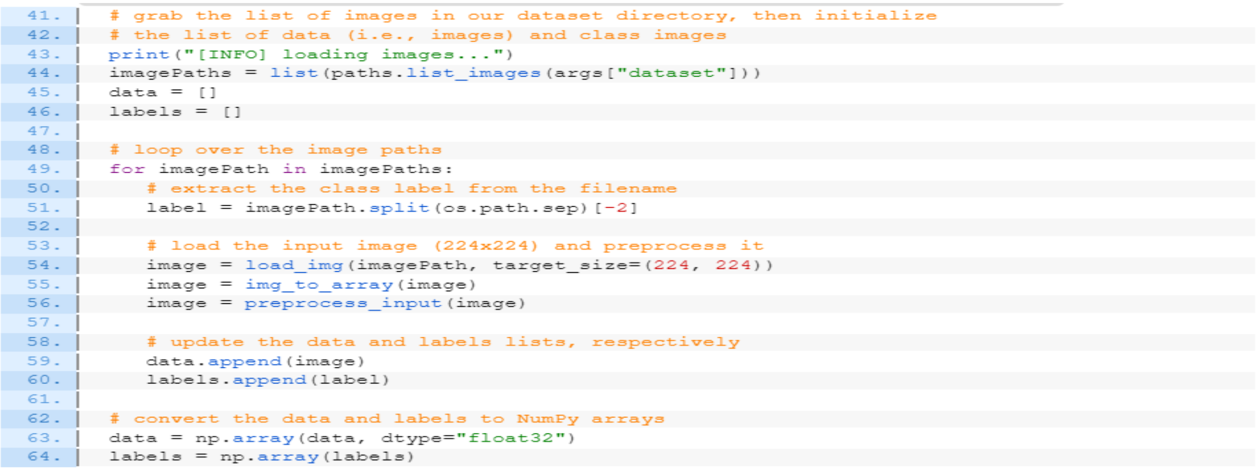
I like to define my deep learning hyperparameters in one place:



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

Here, I’ve specified hyperparameter constants including my initial learning rate, number of training epochs, and batch size. Later, we will be applying a learning rate decay schedule, which is why we’ve named the learning rate variable INIT_LR.

At this point, we’re ready to load and pre-process our training data:



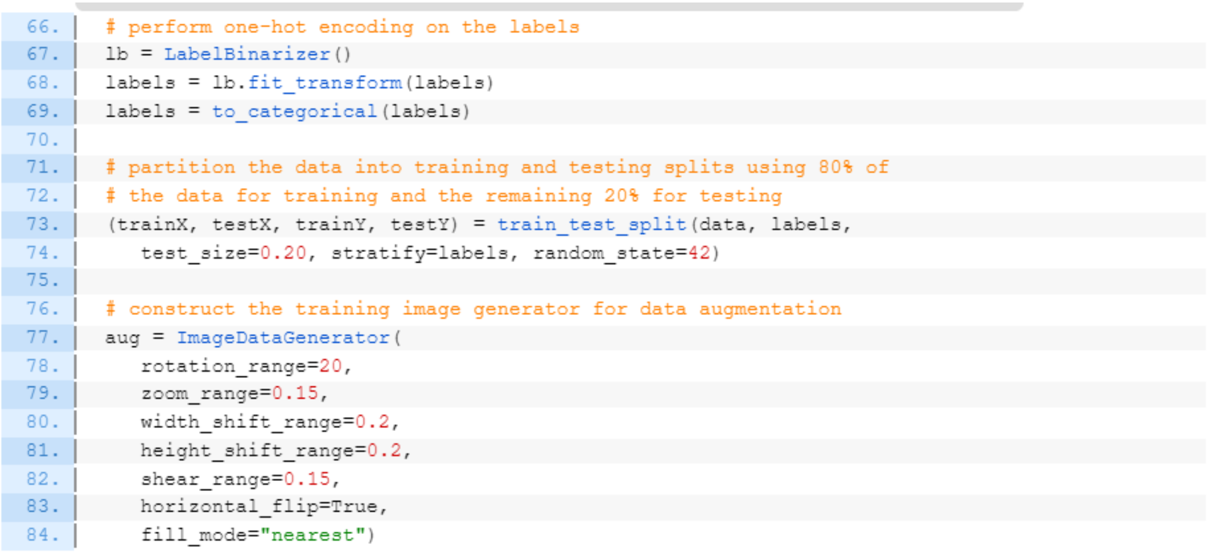
```
# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# 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)
```

In this block, we are:

- Grabbing all of the imagePaths in the dataset (Line 44)
- Initializing data and labels lists (Lines 45 and 46)
- Looping over the imagePaths and loading + pre-processing images (Lines 49-60). Pre-processing steps include resizing to 224×224 pixels, conversion to array format, and scaling the pixel intensities in the input image to the range [-1, 1] (via the preprocess_input convenience function)
- Appending the pre-processed image and associated label to the data and labels lists, respectively (Lines 59 and 60)
- Ensuring our training data is in NumPy array format (Lines 63 and 64)

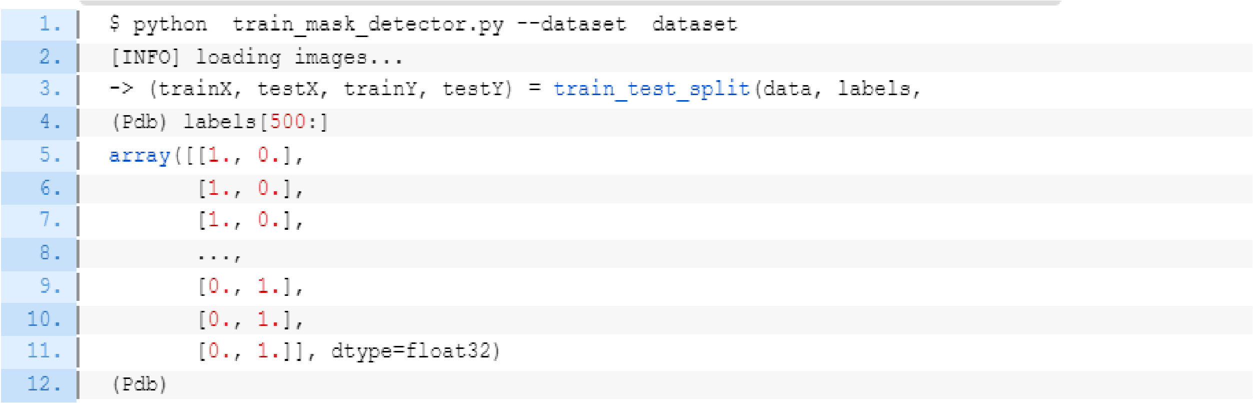
The above lines of code assume that your entire dataset is small enough to fit into memory. If your dataset is larger than the memory you have available, I suggest using HDF5

Our data preparation work isn't done yet. Next, we'll encode our labels, partition our dataset, and prepare for data augmentation:



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

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(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")
```

Lines 67-69 one-hot encode our class labels, meaning that our data will be in the following format:



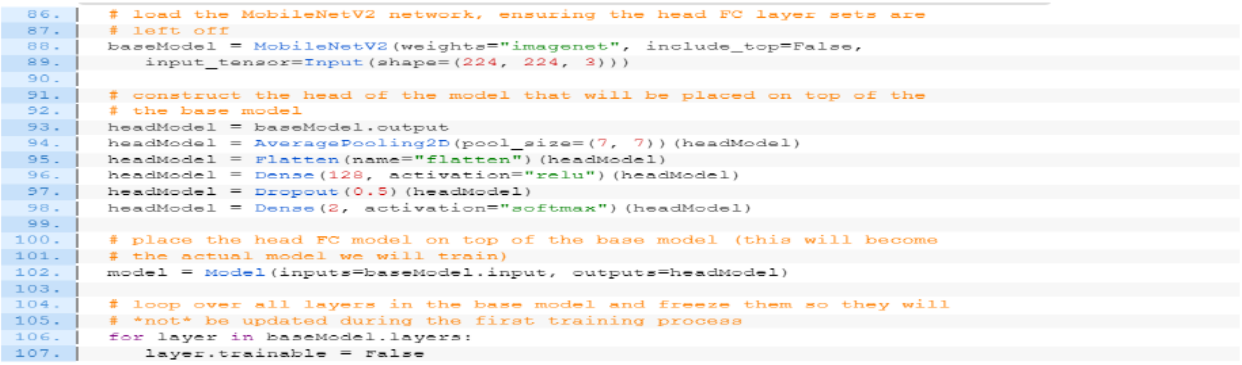
```
$ python  train_mask_detector.py --dataset  dataset
[INFO] loading images...
-> (trainX, testX, trainY, testY) = train_test_split(data, labels,
(Pdb) labels[500:]
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
(Pdb)
```

As you can see, each element of our labels array consists of an array in which only one index is “hot” (i.e., 1).

Using scikit-learn’s convenience method, Lines 73 and 74 segment our data into 80% training and the remaining 20% for testing.

During training, we’ll be applying on-the-fly mutations to our images in an effort to improve generalization. This is known as data augmentation, where the random rotation, zoom, shear, shift, and flip parameters are established on Lines 77-84. We’ll use the aug object at training time.

But first, we need to prepare MobileNetV2 for fine-tuning:



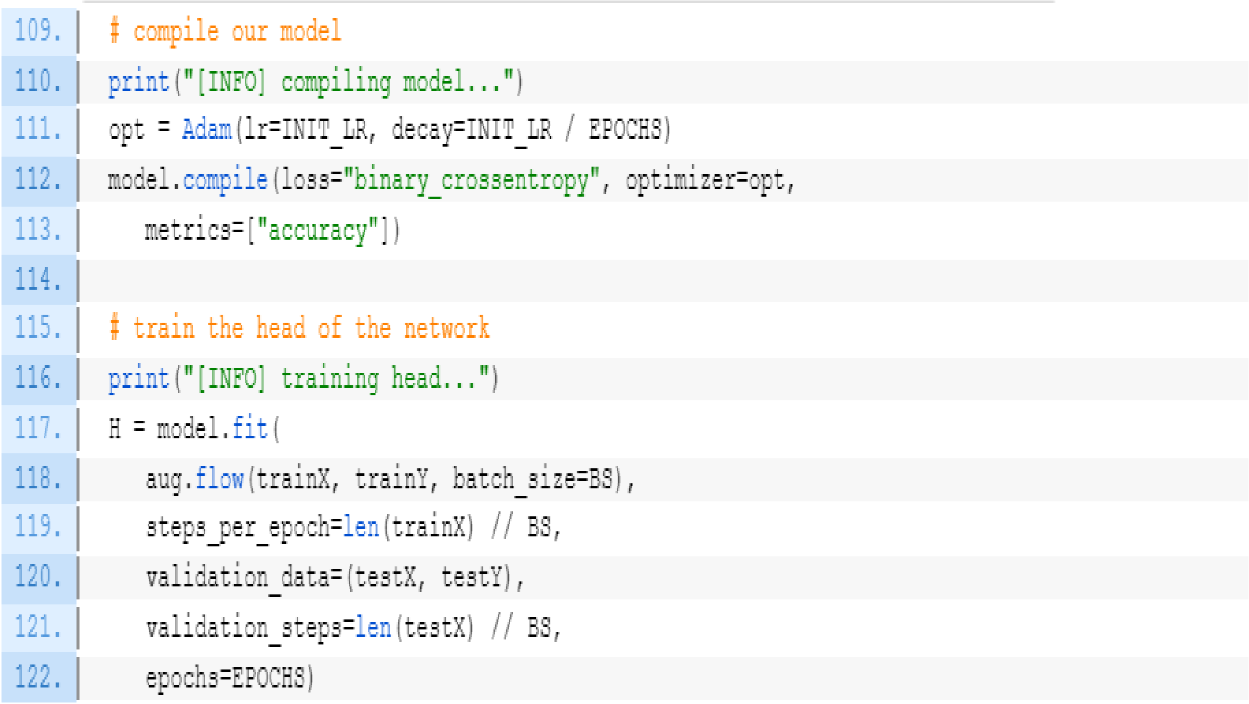
```
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

# 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
```

Fine-tuning setup is a three-step process:

- Load MobileNet with pre-trained ImageNet weights, leaving off head of network (Lines 88 and 89)
- Construct a new FC head, and append it to the base in place of the old head (Lines 93-102)
- Freeze the base layers of the network (Lines 106 and 107). The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights will be tuned.

Fine-tuning is a strategy I nearly always recommend to establish a baseline model while saving considerable time.

With our data prepared and model architecture in place for fine-tuning, we're now ready to compile and train our face mask detector network:

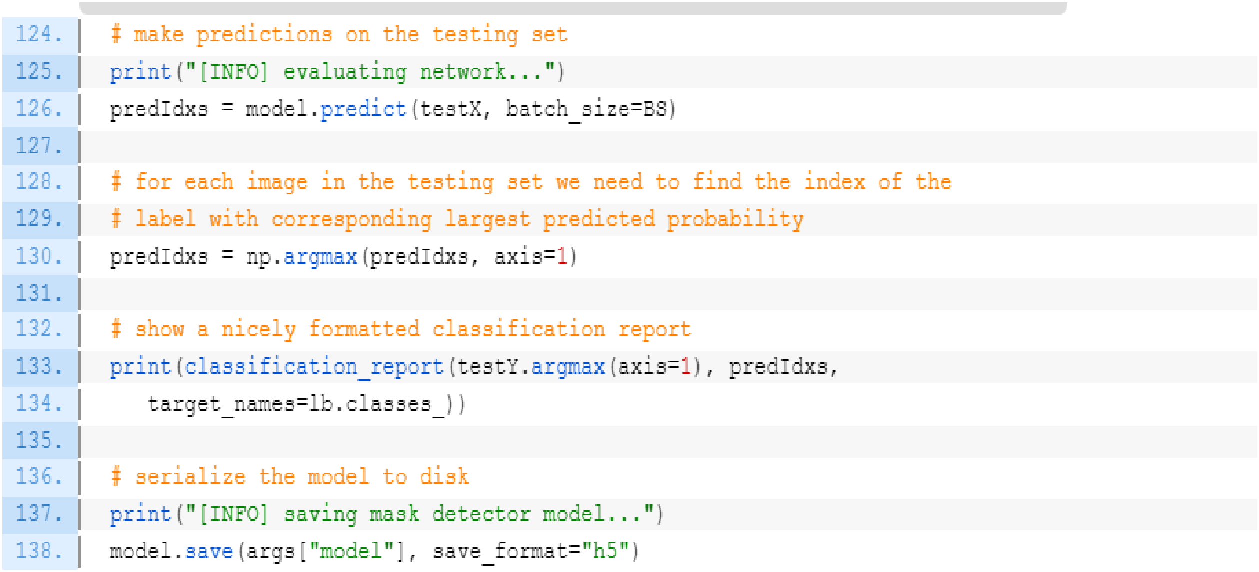


```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# 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)
```

Lines 111-113 compile our model with the Adam optimizer, a learning rate decay schedule, and binary cross-entropy. If you're building from this training script with > 2 classes, be sure to use categorical cross-entropy.

Face mask training is launched via Lines 117-122. Notice how our data augmentation object (aug) will be providing batches of mutated image data.

Once training is complete, we'll evaluate the resulting model on the test set:

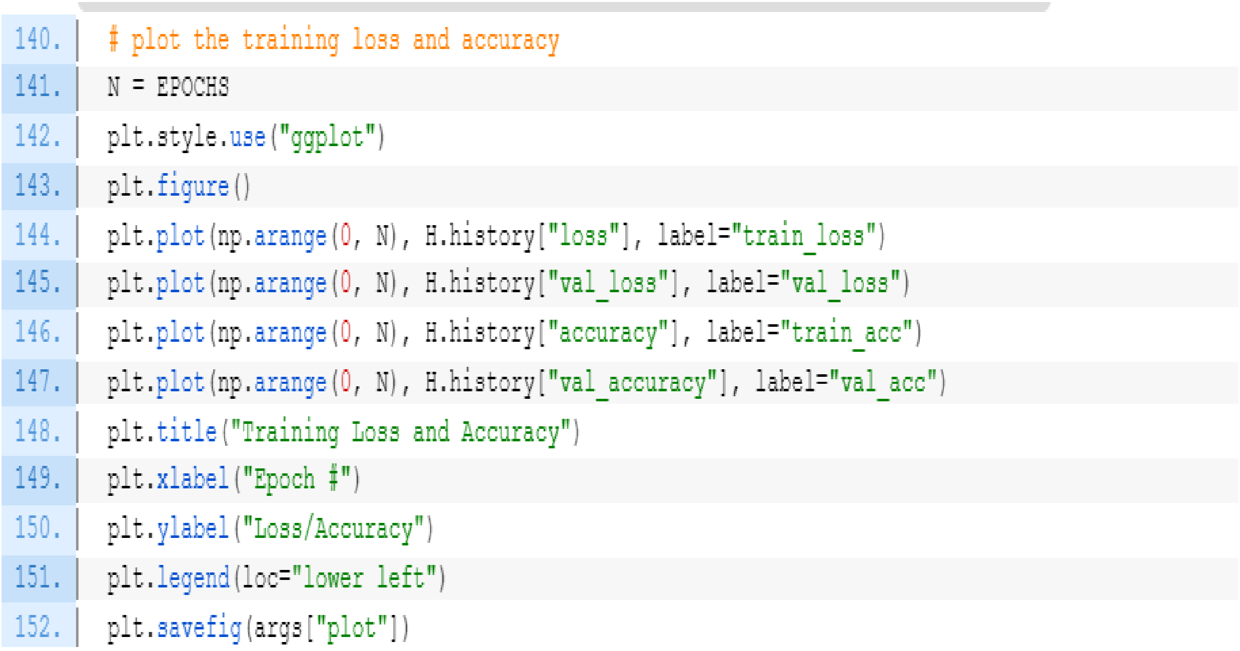


```
124. # make predictions on the testing set
125. print("[INFO] evaluating network...")
126. predIdxs = model.predict(testX, batch_size=BS)
127.
128. # for each image in the testing set we need to find the index of the
129. # label with corresponding largest predicted probability
130. predIdxs = np.argmax(predIdxs, axis=1)
131.
132. # show a nicely formatted classification report
133. print(classification_report(testY.argmax(axis=1), predIdxs,
134.     target_names=lb.classes_))
135.
136. # serialize the model to disk
137. print("[INFO] saving mask detector model...")
138. model.save(args["model"], save_format="h5")
```

Here, Lines 126-130 make predictions on the test set, grabbing the highest probability class label indices. Then, we print a classification report in the terminal for inspection.

Line 138 serializes our face mask classification model to disk.

Our last step is to plot our accuracy and loss curves:

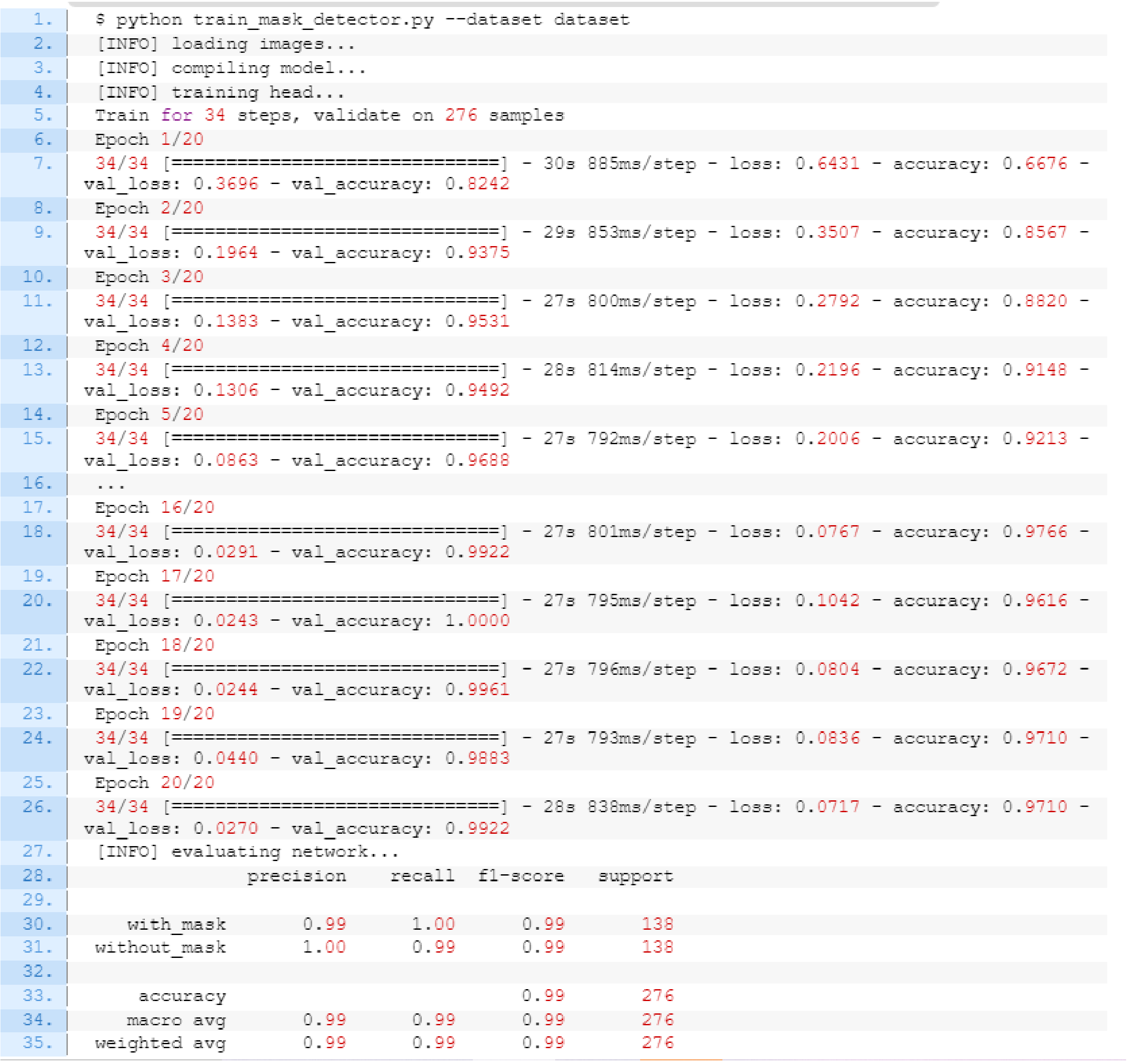


```
# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 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(args["plot"])
```

Training the COVID-19 face mask detector with Keras/TensorFlow

We are now ready to train our face mask detector using Keras, TensorFlow, and Deep Learning.

From there, open up a terminal, and execute the following command:



```
$ python train_mask_detector.py --dataset dataset
[INFO] loading images...
[INFO] compiling model...
[INFO] training head...
Train for 34 steps, validate on 276 samples
Epoch 1/20
34/34 [==============================] - 30s 885ms/step - loss: 0.6431 - accuracy: 0.6676 - val_loss: 0.3696 - val_accuracy: 0.8242
Epoch 2/20
34/34 [==============================] - 29s 853ms/step - loss: 0.3507 - accuracy: 0.8567 - val_loss: 0.1964 - val_accuracy: 0.9375
Epoch 3/20
34/34 [==============================] - 27s 800ms/step - loss: 0.2792 - accuracy: 0.8820 - val_loss: 0.1383 - val_accuracy: 0.9531
Epoch 4/20
34/34 [==============================] - 28s 814ms/step - loss: 0.2196 - accuracy: 0.9148 - val_loss: 0.1306 - val_accuracy: 0.9492
Epoch 5/20
34/34 [==============================] - 27s 792ms/step - loss: 0.2006 - accuracy: 0.9213 - val_loss: 0.0863 - val_accuracy: 0.9688
...
Epoch 16/20
34/34 [==============================] - 27s 801ms/step - loss: 0.0767 - accuracy: 0.9766 - val_loss: 0.0291 - val_accuracy: 0.9922
Epoch 17/20
34/34 [==============================] - 27s 795ms/step - loss: 0.1042 - accuracy: 0.9616 - val_loss: 0.0243 - val_accuracy: 1.0000
Epoch 18/20
34/34 [==============================] - 27s 796ms/step - loss: 0.0804 - accuracy: 0.9672 - val_loss: 0.0244 - val_accuracy: 0.9961
Epoch 19/20
34/34 [==============================] - 27s 793ms/step - loss: 0.0836 - accuracy: 0.9710 - val_loss: 0.0440 - val_accuracy: 0.9883
Epoch 20/20
34/34 [==============================] - 28s 838ms/step - loss: 0.0717 - accuracy: 0.9710 - val_loss: 0.0270 - val_accuracy: 0.9922
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       0.99      1.00      0.99       138
without_mask       1.00      0.99      0.99       138

    accuracy                           0.99       276
   macro avg       0.99      0.99      0.99       276
weighted avg       0.99      0.99      0.99       276
```

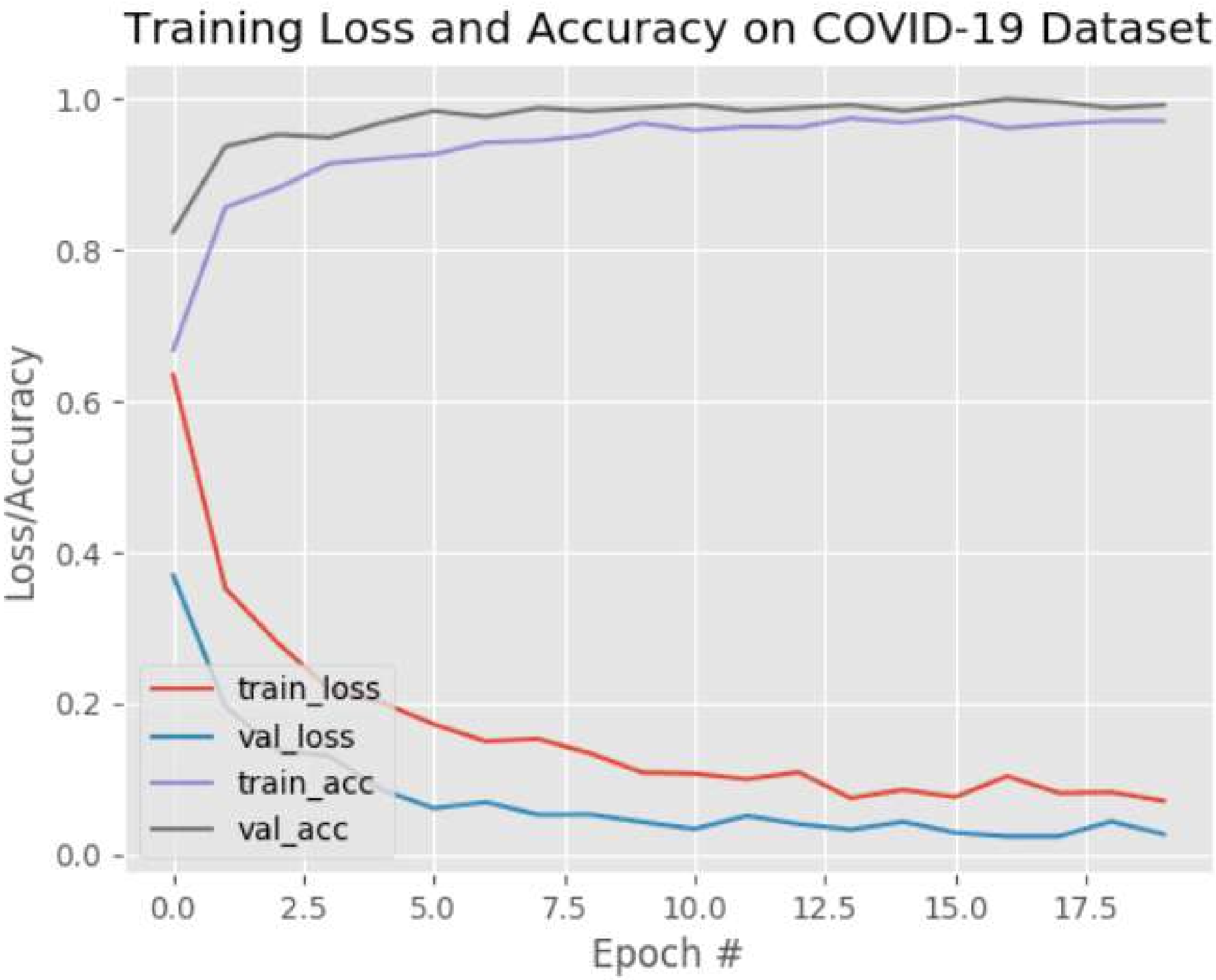


Fig. COVID-19 face mask detector training accuracy/loss curves demonstrate high accuracy and little signs of overfitting on the data. We're now ready to apply our knowledge of computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras to perform face mask detection.

As you can see, we are obtaining ~99% accuracy on our test set.

Looking at Figure 10, we can see there are little signs of overfitting, with the validation loss lower than the training loss.

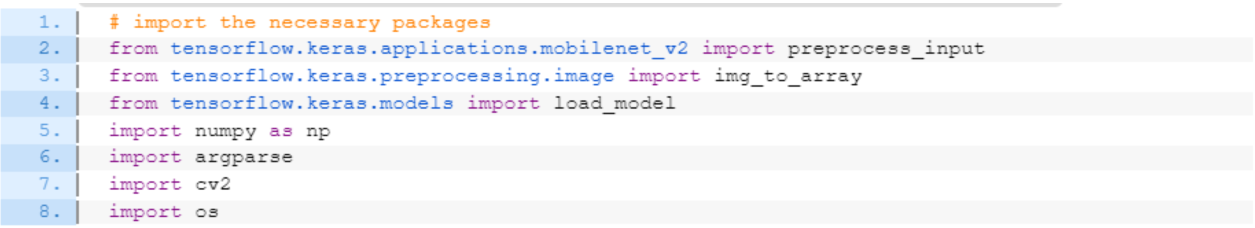
Given these results, we are hopeful that our model will generalize well to images outside our training and testing set.

Implementing our COVID-19 face mask detector for images with OpenCV

Now that our face mask detector is trained, let’s learn how we can:

- Load an input image from disk
- Detect faces in the image
- Apply our face mask detector to classify the face as either with_mask or without_mask

Open up the detect_mask_image.py file in your directory structure, and let’s get started:

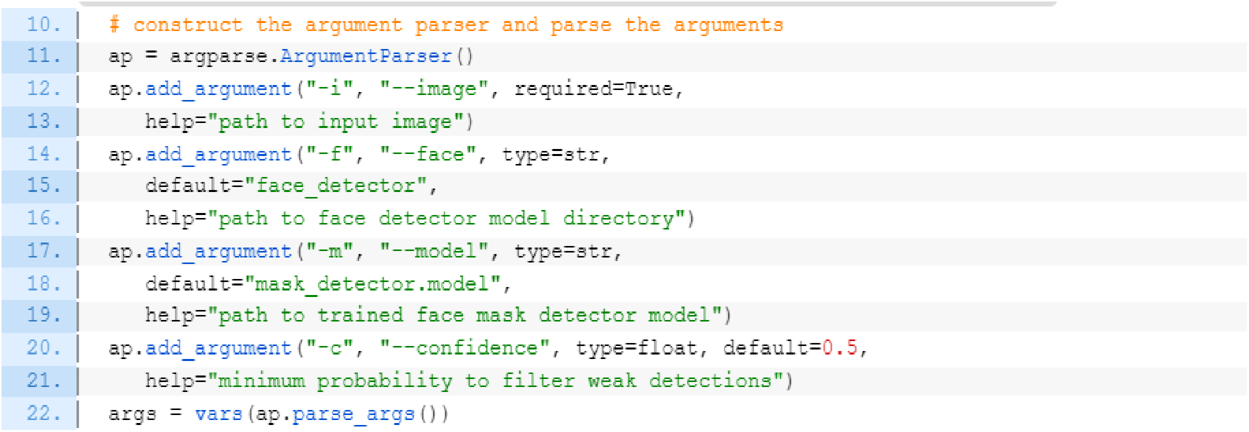


```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import cv2
import os
```

Our driver script requires three TensorFlow/Keras imports to (1) load our MaskNet model and (2) pre-process the input image.

OpenCV is required for display and image manipulations.

The next step is to parse command line arguments:

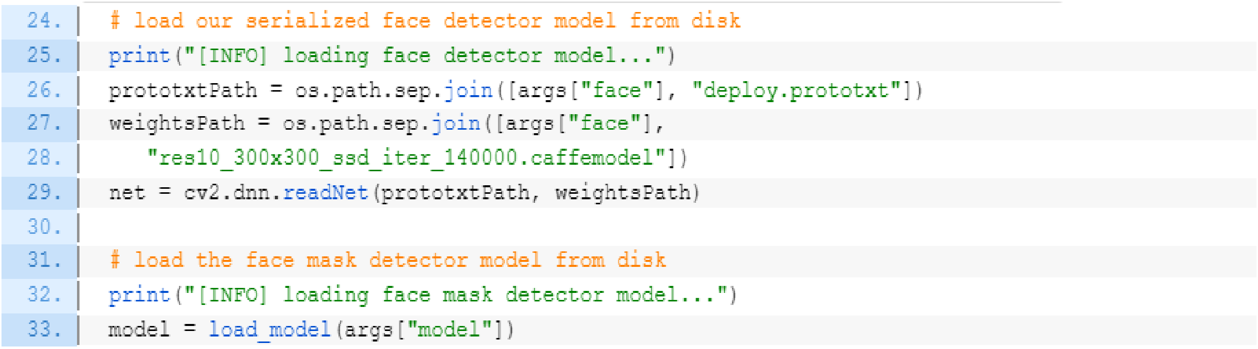


```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
   help="path to input image")
ap.add_argument("-f", "--face", type=str,
   default="face_detector",
   help="path to face detector model directory")
ap.add_argument("-m", "--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())
```

Our four command line arguments include:

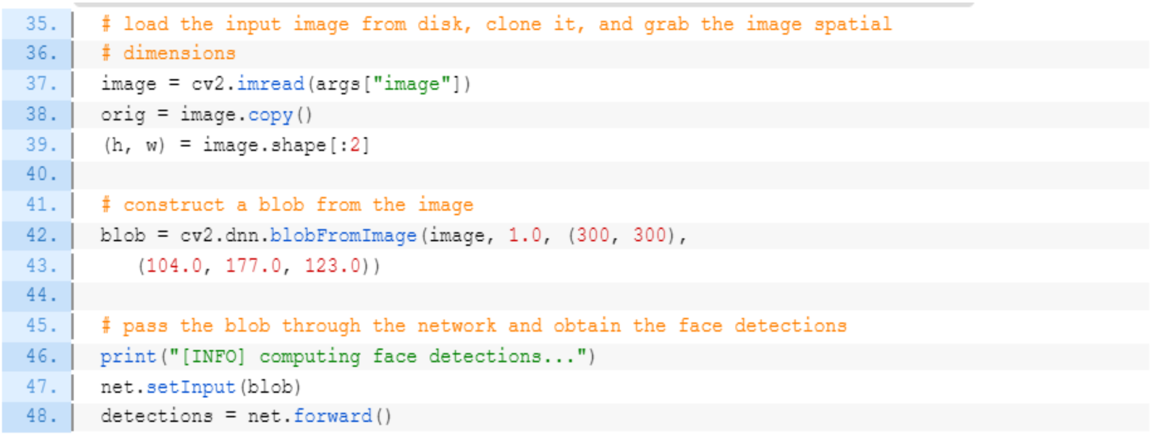
- --image: The path to the input image containing faces for inference
- --face: The path to the face detector model directory (we need to localize faces prior to classifying them)
- --model: The path to the face mask detector model that we trained earlier in this tutorial
- --confidence: An optional probability threshold can be set to override 50% to filter weak face detections

Next, we'll load both our face detector and face mask classifier models:



```
# load our serialized face detector model from disk
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"])
net = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
model = load_model(args["model"])
```

With our deep learning models now in memory, our next step is to load and pre-process an input image:



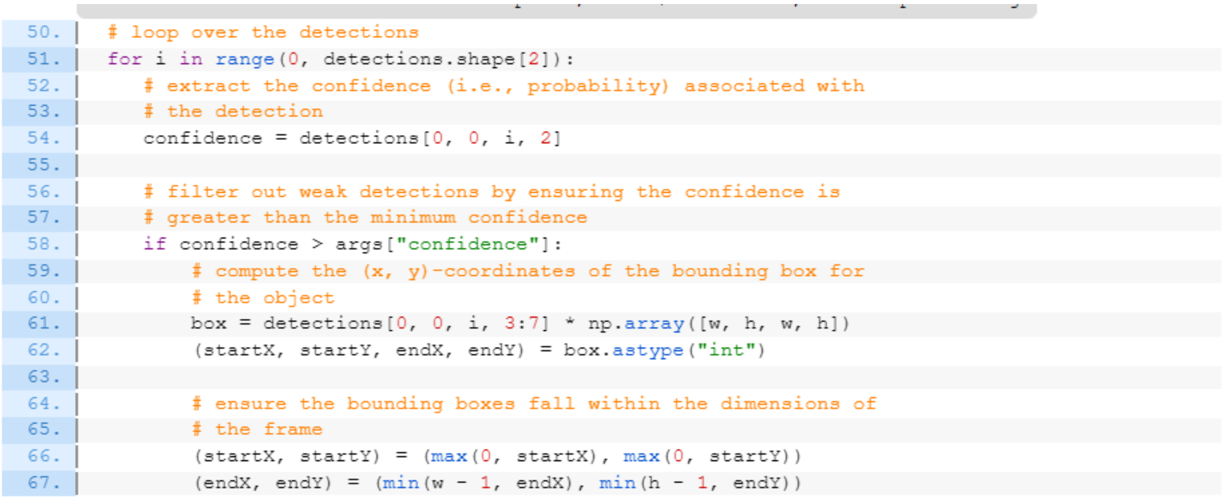
```
# load the input image from disk, clone it, and grab the image spatial
# dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(h, w) = image.shape[:2]

# construct a blob from the image
blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
   (104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections
print("[INFO] computing face detections...")
net.setInput(blob)
detections = net.forward()
```

Upon loading our --image from disk (Line 37), we make a copy and grab frame dimensions for future scaling and display purposes (Lines 38 and 39).

Pre-processing is handled by OpenCV's blobFromImage function (Lines 42 and 43). As shown in the parameters, we resize to 300×300 pixels and perform mean subtraction.

Lines 47 and 48 then perform face detection to localize where in the image all faces are.

Once we know where each face is predicted to be, we'll ensure they meet the --confidence threshold before we extract the faceROIs:

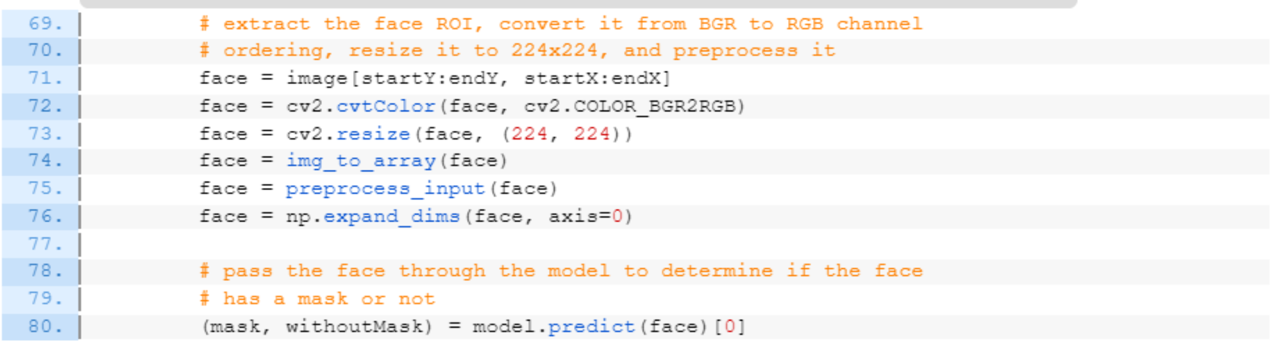


```
50. # loop over the detections
51. for i in range(0, detections.shape[2]):
52.     # extract the confidence (i.e., probability) associated with
53.     # the detection
54.     confidence = detections[0, 0, i, 2]
55.
56.     # filter out weak detections by ensuring the confidence is
57.     # greater than the minimum confidence
58.     if confidence > args["confidence"]:
59.         # compute the (x, y)-coordinates of the bounding box for
60.         # the object
61.         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
62.         (startX, startY, endX, endY) = box.astype("int")
63.
64.         # ensure the bounding boxes fall within the dimensions of
65.         # the frame
66.         (startX, startY) = (max(0, startX), max(0, startY))
67.         (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Here, we loop over our detections and extract the confidence to measure against the --confidence threshold (Lines 51-58).

We then compute the bounding box value for a particular face and ensure that the box falls within the boundaries of the image (Lines 61-67).

Next, we'll run the face ROI through our MaskNet model:

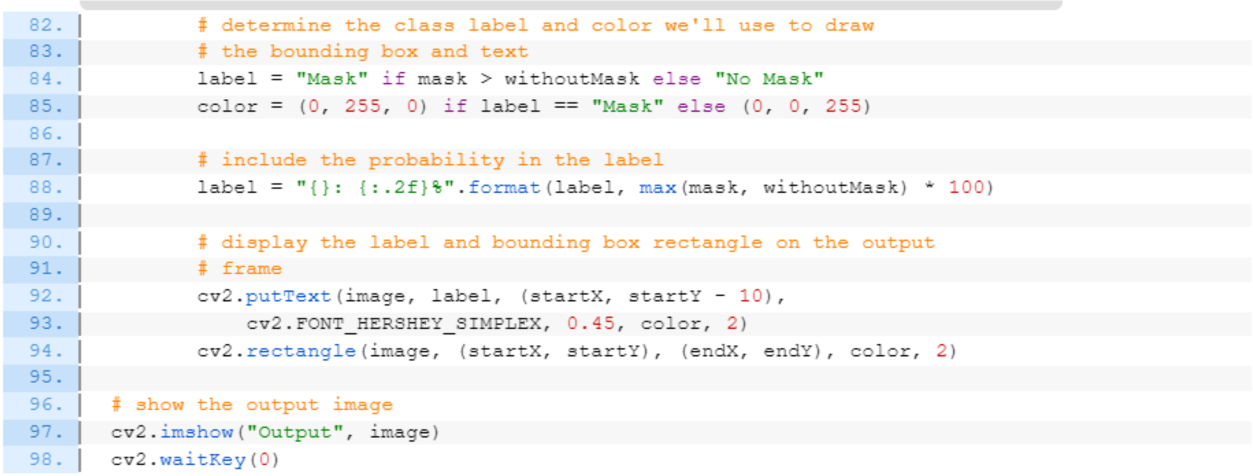


```
69.         # extract the face ROI, convert it from BGR to RGB channel
70.         # ordering, resize it to 224x224, and preprocess it
71.         face = image[startY:endY, startX:endX]
72.         face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
73.         face = cv2.resize(face, (224, 224))
74.         face = img_to_array(face)
75.         face = preprocess_input(face)
76.         face = np.expand_dims(face, axis=0)
77.
78.         # pass the face through the model to determine if the face
79.         # has a mask or not
80.         (mask, withoutMask) = model.predict(face)[0]
```

In this block, we:

- Extract the face ROI via NumPy slicing (Line 71)
- Pre-process the ROI the same way we did during training (Lines 72-76)
- Perform mask detection to predict with_mask or without_mask (Line 80)

From here, we will annotate and display the result!



```
82.         # determine the class label and color we'll use to draw
83.         # the bounding box and text
84.         label = "Mask" if mask > withoutMask else "No Mask"
85.         color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
86.
87.         # include the probability in the label
88.         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
89.
90.         # display the label and bounding box rectangle on the output
91.         # frame
92.         cv2.putText(image, label, (startX, startY - 10),
93.             cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
94.         cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
95.
96. # show the output image
97. cv2.imshow("Output", image)
98. cv2.waitKey(0)
```

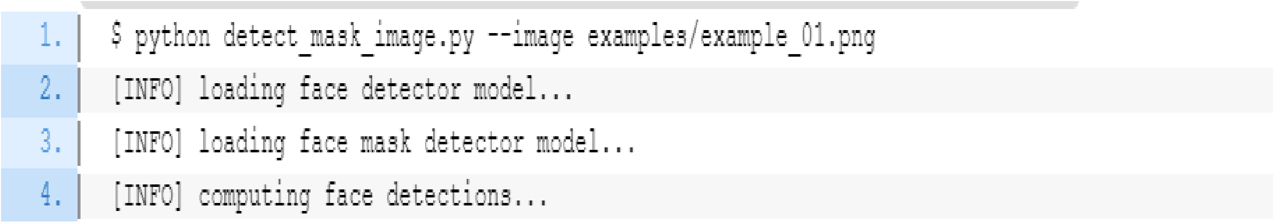
First, we determine the class label based on probabilities returned by the mask detector model (Line 84) and assign an associated color for the annotation (Line 85). The color will be "green" for with_mask and "red" for without_mask.

We then draw the label text (including class and probability), as well as a bounding box rectangle for the face, using OpenCV drawing functions (Lines 92-94).

Once all detections have been processed, Lines 97 and 98 display the output image.

COVID-19 face mask detection in images with OpenCV

From there, open up a terminal, and execute the following command:



```
$ python detect_mask_image.py --image examples/example_01.png
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] computing face detections...
```

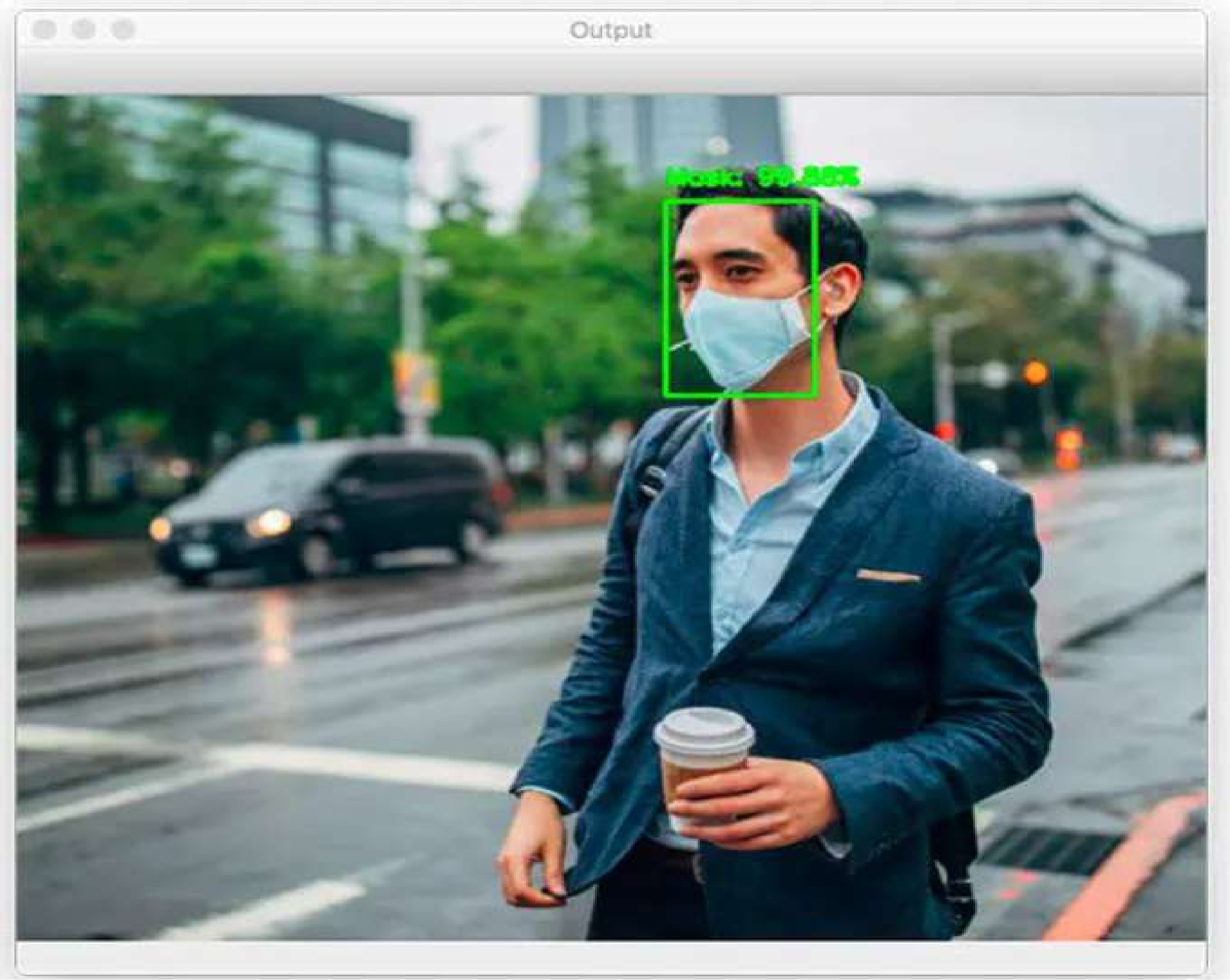
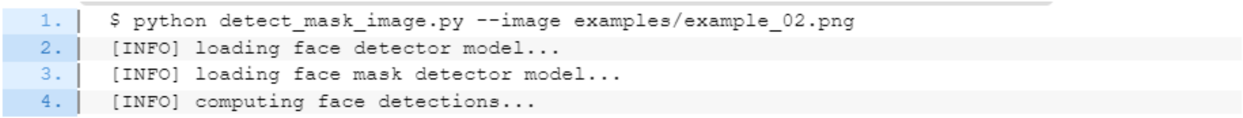


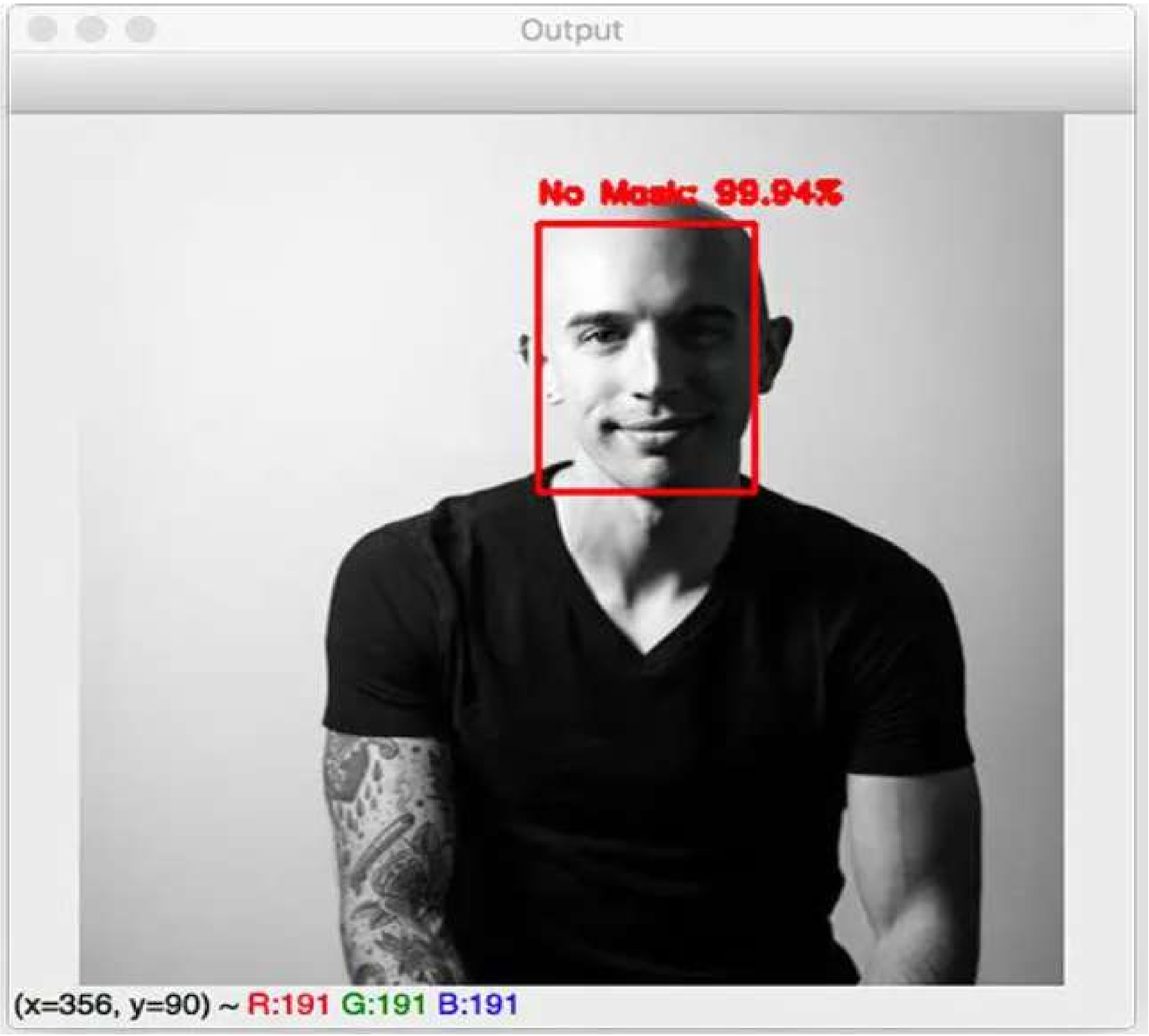
Fig Is this man wearing a COVID-19/Coronavirus face mask in public? Yes, he is and our computer vision and deep learning method using Python, OpenCV, and TensorFlow/Keras has made it possible to detect the presence of the mask automatically.

As you can see, our face mask detector correctly labeled this image as Mask.

Let’s try another image, this one of a person not wearing a face mask:



```
$ python detect_mask_image.py --image examples/example_02.png
[INFO] loading face detector model...
[INFO] loading face mask detector model...
[INFO] computing face detections...
```



```
(x=356, y=90) ~ R:191 G:191 B:191
```

Fig. Uh oh. I’m not wearing a COVID-19 face mask in this picture. Using Python, OpenCV, and TensorFlow/Keras, our system has correctly detected “No Mask” for my face.

Our face mask detector has correctly predicted No Mask.

Let’s try one final image:

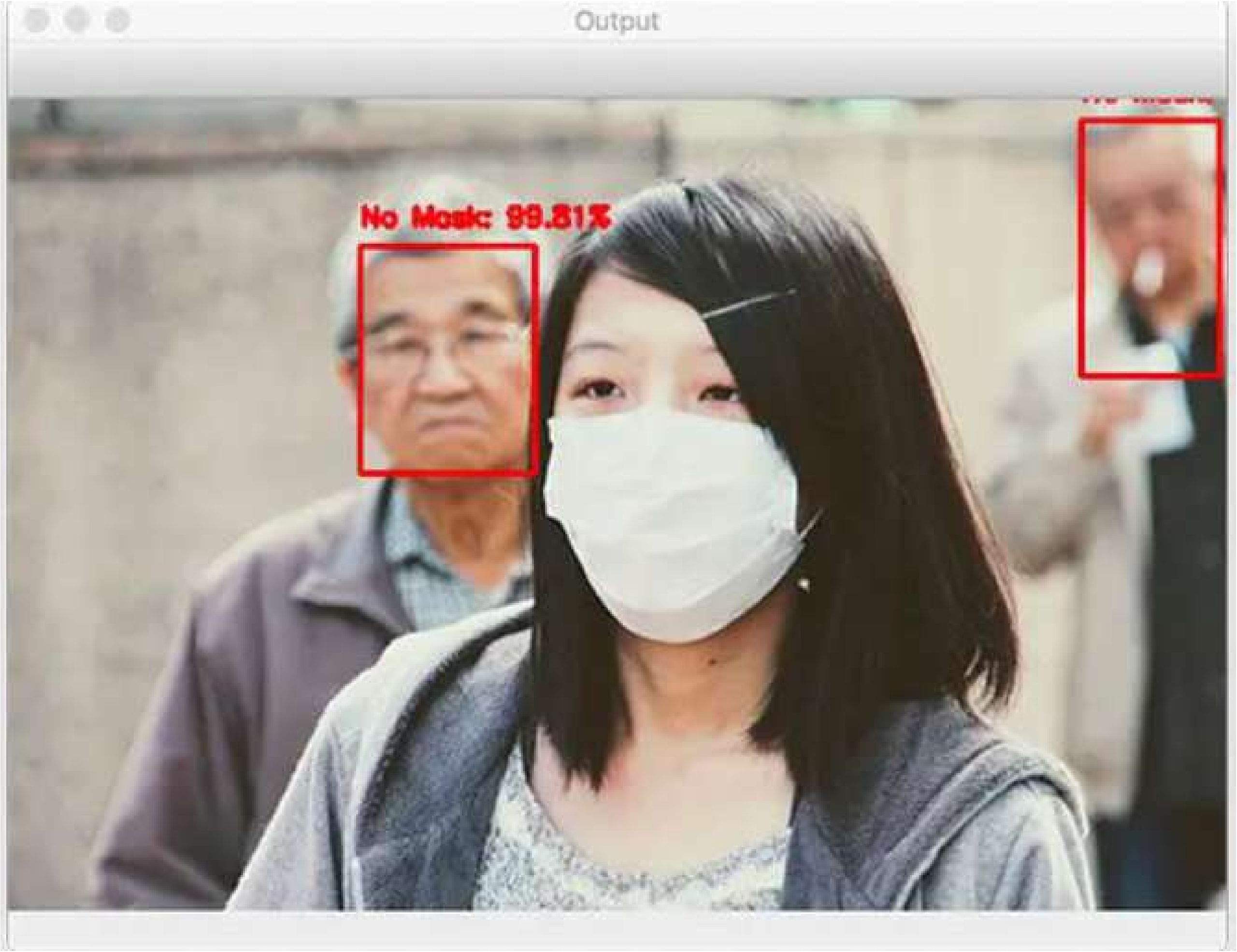


Fig. What is going on in this result? Why is the lady in the foreground not detected as wearing a COVID-19 face mask? Has our COVID-19 face mask detector built with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras failed us?

What happened here?

Why is it that we were able to detect the faces of the two gentlemen in the background and correctly classify mask/no mask for them, but we could not detect the woman in the foreground?

The gist is that we’re too reliant on our two-stage process.

Keep in mind that in order to classify whether or not a person is wearing in mask, we first need to perform face detection — if a face is not found (which is what happened in this image), then the mask detector cannot be applied!

The reason we cannot detect the face in the foreground is because:

It’s too obscured by the mask
The dataset used to train the face detector did not contain example images of people wearing face masks
Therefore, if a large portion of the face is occluded, our face detector will likely fail to detect the face.

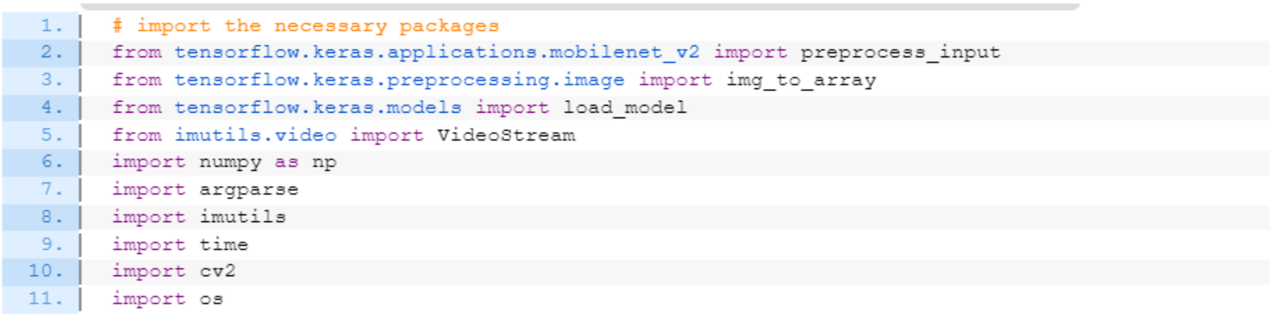
Implementing our COVID-19 face mask detector in real-time video streams with OpenCV

At this point, we know we can apply face mask detection to static images — but what about real-time video streams?

Is our COVID-19 face mask detector capable of running in real-time?

Let’s find out.

Open up the detect_mask_video.py file in your directory structure, and insert the following code:

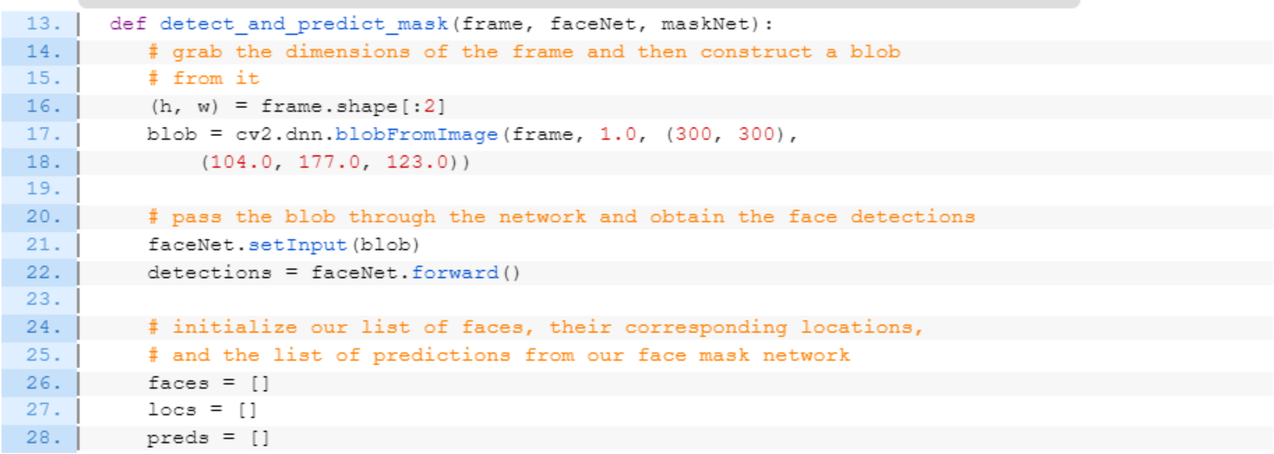


```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils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

The algorithm for this script is the same, but it is pieced together in such a way to allow for processing every frame of your webcam stream.

Thus, the only difference when it comes to imports is that we need a VideoStream class and time. Both of these will help us to work with the stream. We’ll also take advantage of imutils for its aspect-aware resizing method.

Our face detection/mask prediction logic for this script is in the detect_and_predict_mask function:



```
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, (300, 300),
        (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 = []
```

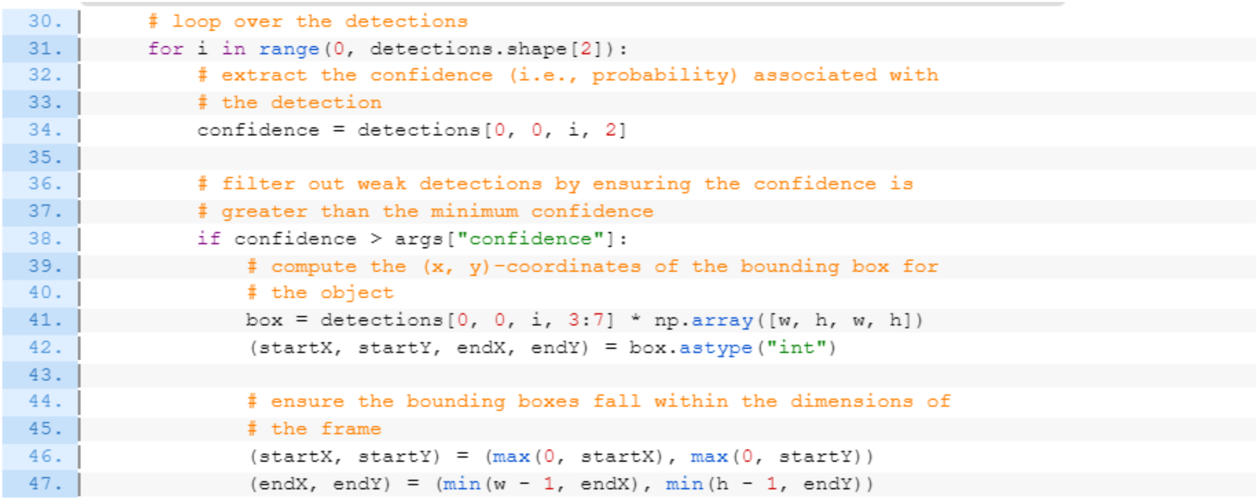
By defining this convenience function here, our frame processing loop will be a little easier to read later.

This function detects faces and then applies our face mask classifier to each face ROI. Such a function consolidates our code — it could even be moved to a separate Python file if you so choose.

Our detect_and_predict_mask function accepts three parameters:

- frame: A frame from our stream
- faceNet: The model used to detect where in the image faces are
- maskNet: Our COVID-19 face mask classifier model
- Inside, we construct a blob, detect faces, and initialize lists, two of which the function is set to return. These lists include our faces (i.e., ROIs), locs (the face locations), and preds (the list of mask/no mask predictions).

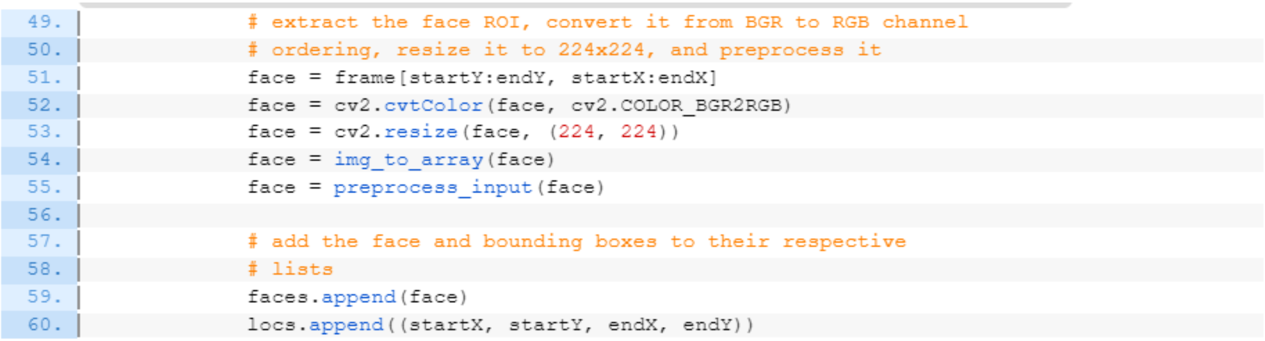
From here, we'll loop over the face detections:



```
30.     # loop over the detections
31.     for i in range(0, detections.shape[2]):
32.         # extract the confidence (i.e., probability) associated with
33.         # the detection
34.         confidence = detections[0, 0, i, 2]
35.
36.         # filter out weak detections by ensuring the confidence is
37.         # greater than the minimum confidence
38.         if confidence > args["confidence"]:
39.             # compute the (x, y)-coordinates of the bounding box for
40.             # the object
41.             box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
42.             (startX, startY, endX, endY) = box.astype("int")
43.
44.             # ensure the bounding boxes fall within the dimensions of
45.             # the frame
46.             (startX, startY) = (max(0, startX), max(0, startY))
47.             (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

Inside the loop, we filter out weak detections (Lines 34-38) and extract bounding boxes while ensuring bounding box coordinates do not fall outside the bounds of the image (Lines 41-47).

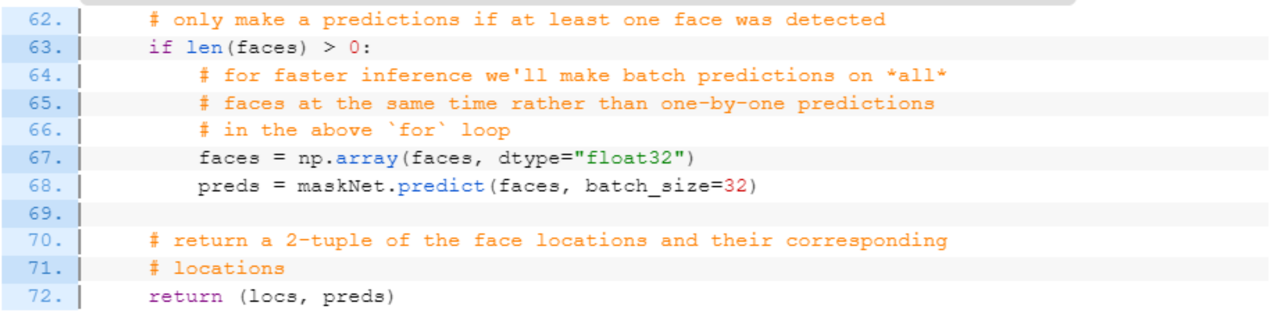
Next, we'll add face ROIs to two of our corresponding lists:



```
49.             # extract the face ROI, convert it from BGR to RGB channel
50.             # ordering, resize it to 224x224, and preprocess it
51.             face = frame[startY:endY, startX:endX]
52.             face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
53.             face = cv2.resize(face, (224, 224))
54.             face = img_to_array(face)
55.             face = preprocess_input(face)
56.
57.             # add the face and bounding boxes to their respective
58.             # lists
59.             faces.append(face)
60.             locs.append((startX, startY, endX, endY))
```

After extracting face ROIs and pre-processing (Lines 51-56), we append the the face ROIs and bounding boxes to their respective lists.

We're now ready to run our faces through our mask predictor:



```
62.     # only make a predictions if at least one face was detected
63.     if len(faces) > 0:
64.         # for faster inference we'll make batch predictions on *all*
65.         # faces at the same time rather than one-by-one predictions
66.         # in the above `for` loop
67.         faces = np.array(faces, dtype="float32")
68.         preds = maskNet.predict(faces, batch_size=32)
69.
70.     # return a 2-tuple of the face locations and their corresponding
71.     # locations
72.     return (locs, preds)
```

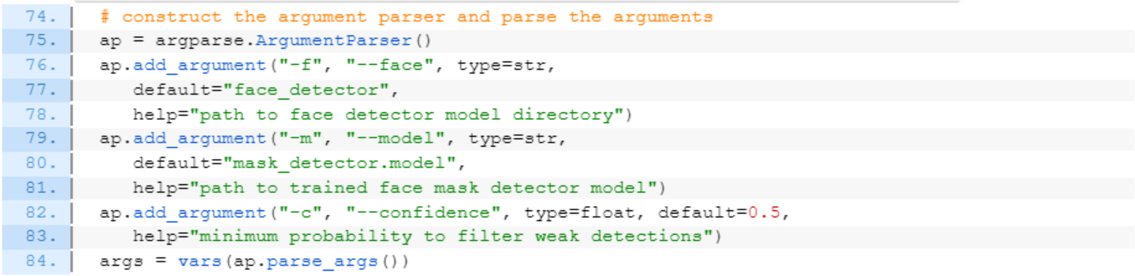
The logic here is built for speed. First we ensure at least one face was detected (Line 63) — if not, we'll return empty preds.

with Line 67 to convert faces into a 32-bit floating point NumPy array. Additionally, Line 61 from the previous block has been removed (formerly, it added an unnecessary batch dimension). The combination of these two changes now fixes a bug that was preventing multiple preds to be returned from inference. With the fix, multiple faces in a single image are properly recognized as having a mask or not having a mask.

Secondly, we are performing inference on our entire batch of faces in the frame so that our pipeline is faster (Line 68). It wouldn't make sense to write another loop to make predictions on each face individually due to the overhead (especially if you are using a GPU that requires a lot of overhead communication on your system bus). It is more efficient to perform predictions in batches.

Line 72 returns our face bounding box locations and corresponding mask/not mask predictions to the caller.

Next, we'll define our command line arguments:

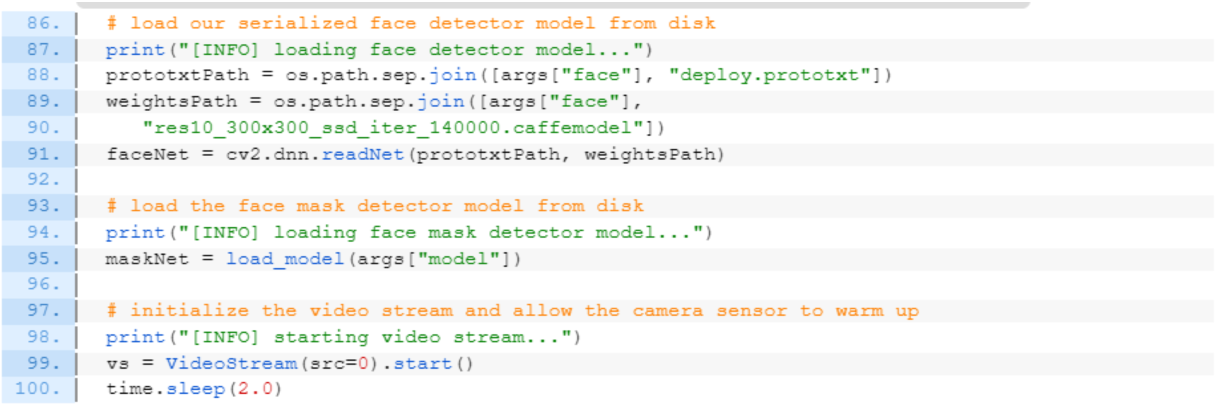


```
74. # construct the argument parser and parse the arguments
75. ap = argparse.ArgumentParser()
76. ap.add_argument("-f", "--face", type=str,
77.     default="face_detector",
78.     help="path to face detector model directory")
79. ap.add_argument("-m", "--model", type=str,
80.     default="mask_detector.model",
81.     help="path to trained face mask detector model")
82. ap.add_argument("-c", "--confidence", type=float, default=0.5,
83.     help="minimum probability to filter weak detections")
84. args = vars(ap.parse_args())
```

Our command line arguments include:

- --face: The path to the face detector directory
- --model: The path to our trained face mask classifier
- --confidence: The minimum probability threshold to filter weak face detections

With our imports, convenience function, and command line args ready to go, we just have a few initializations to handle before we loop over frames:

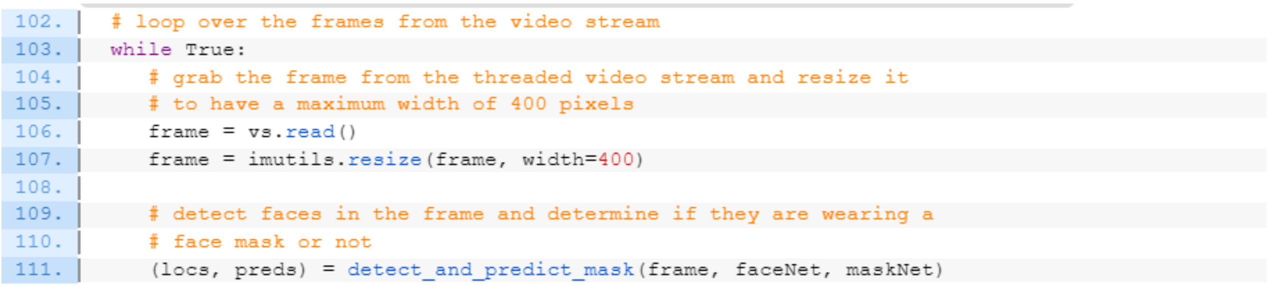


```
86.   # load our serialized face detector model from disk
87.   print("[INFO] loading face detector model...")
88.   prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
89.   weightsPath = os.path.sep.join([args["face"],
90.       "res10_300x300_ssd_iter_140000.caffemodel"])
91.   faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
92.
93.   # load the face mask detector model from disk
94.   print("[INFO] loading face mask detector model...")
95.   maskNet = load_model(args["model"])
96.
97.   # initialize the video stream and allow the camera sensor to warm up
98.   print("[INFO] starting video stream...")
99.   vs = VideoStream(src=0).start()
100.  time.sleep(2.0)
```

Here we have initialized our:

- Face detector
- COVID-19 face mask detector
- Webcam video stream

Let’s proceed to loop over frames in the stream:



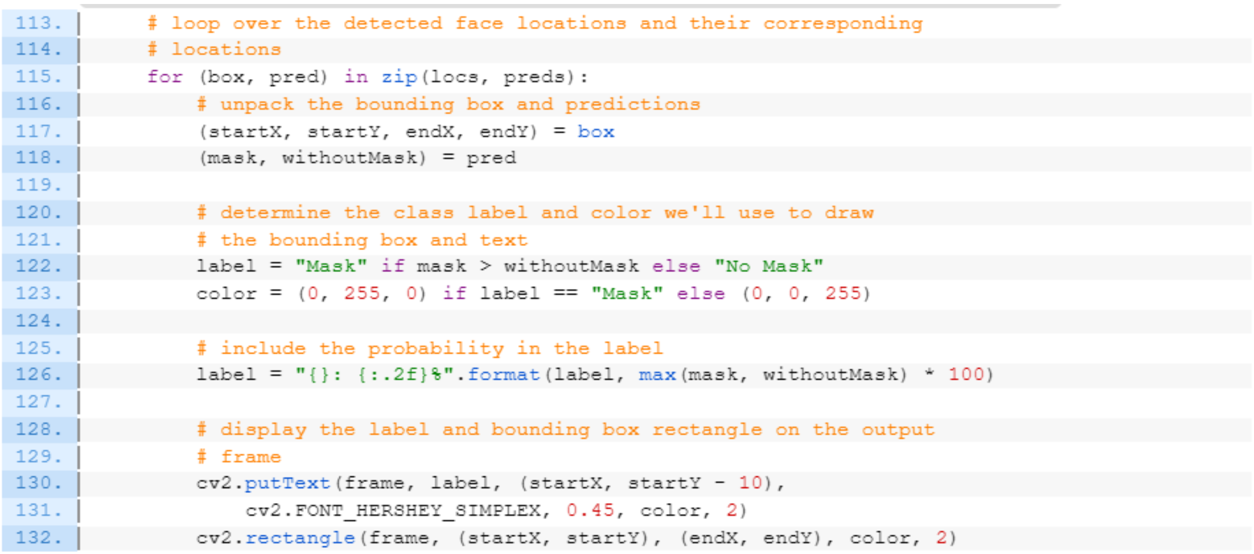
```
102.  # loop over the frames from the video stream
103.  while True:
104.      # grab the frame from the threaded video stream and resize it
105.      # to have a maximum width of 400 pixels
106.      frame = vs.read()
107.      frame = imutils.resize(frame, width=400)
108.
109.      # detect faces in the frame and determine if they are wearing a
110.      # face mask or not
111.      (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
```

We begin looping over frames on Line 103. Inside, we grab a frame from the stream and resize it

(Lines 106 and 107).

From there, we put our convenience utility to use; Line 111 detects and predicts whether people are wearing their masks or not.

Let’s post-process (i.e., annotate) the COVID-19 face mask detection results:

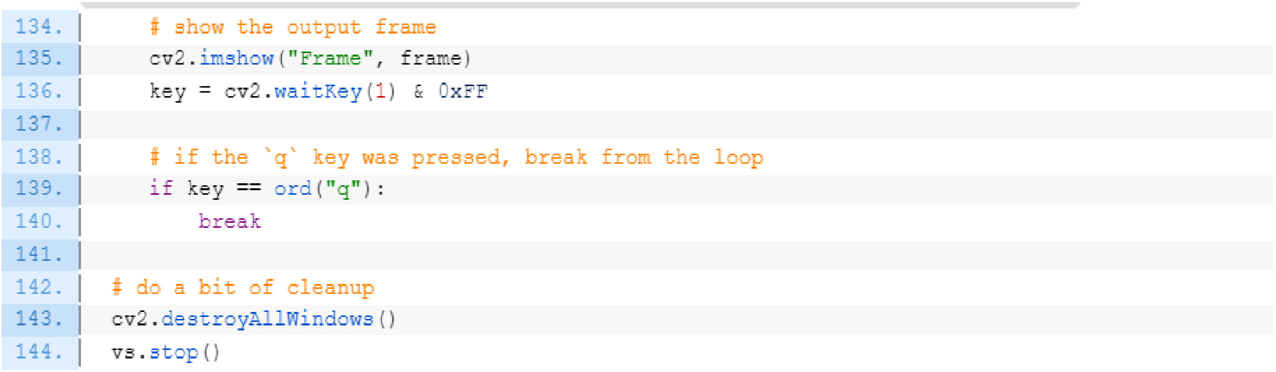


```
113.     # loop over the detected face locations and their corresponding
114.     # locations
115.     for (box, pred) in zip(locs, preds):
116.         # unpack the bounding box and predictions
117.         (startX, startY, endX, endY) = box
118.         (mask, withoutMask) = pred
119. 
120.         # determine the class label and color we'll use to draw
121.         # the bounding box and text
122.         label = "Mask" if mask > withoutMask else "No Mask"
123.         color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
124. 
125.         # include the probability in the label
126.         label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
127. 
128.         # display the label and bounding box rectangle on the output
129.         # frame
130.         cv2.putText(frame, label, (startX, startY - 10),
131.             cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
132.         cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
```

Inside our loop over the prediction results (beginning on Line 115), we:

- Unpack a face bounding box and mask/not mask prediction (Lines 117 and 118)
- Determine the label and color (Lines 122-126)
- Annotate the label and face bounding box (Lines 130-132)

Finally, we display the results and perform cleanup:



```
    # 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()
```

After the frame is displayed, we capture key presses. If the user presses q (quit), we break out of the loop and perform housekeeping.

Great job implementing your real-time face mask detector with Python, OpenCV, and deep learning with TensorFlow/Keras!

To create our face mask detector, we trained a two-class model of people wearing masks and people not wearing masks.

We fine-tuned MobileNetV2 on our mask/no mask dataset and obtained a classifier that is ~99% accurate.

We then took this face mask classifier and applied it to both images and real-time video streams by:

- Detecting faces in images/video
- Extracting each individual face
- Applying our face mask classifier

Our face mask detector is accurate, and since we used the MobileNetV2 architecture, it's also computationally efficient, making it easier to deploy the model to embedded systems (Raspberry Pi, Google Coral, Jetson, Nano, etc.).

Suggestions for improvement

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

Having limited training data

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:

Step #1: Perform face detection

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.

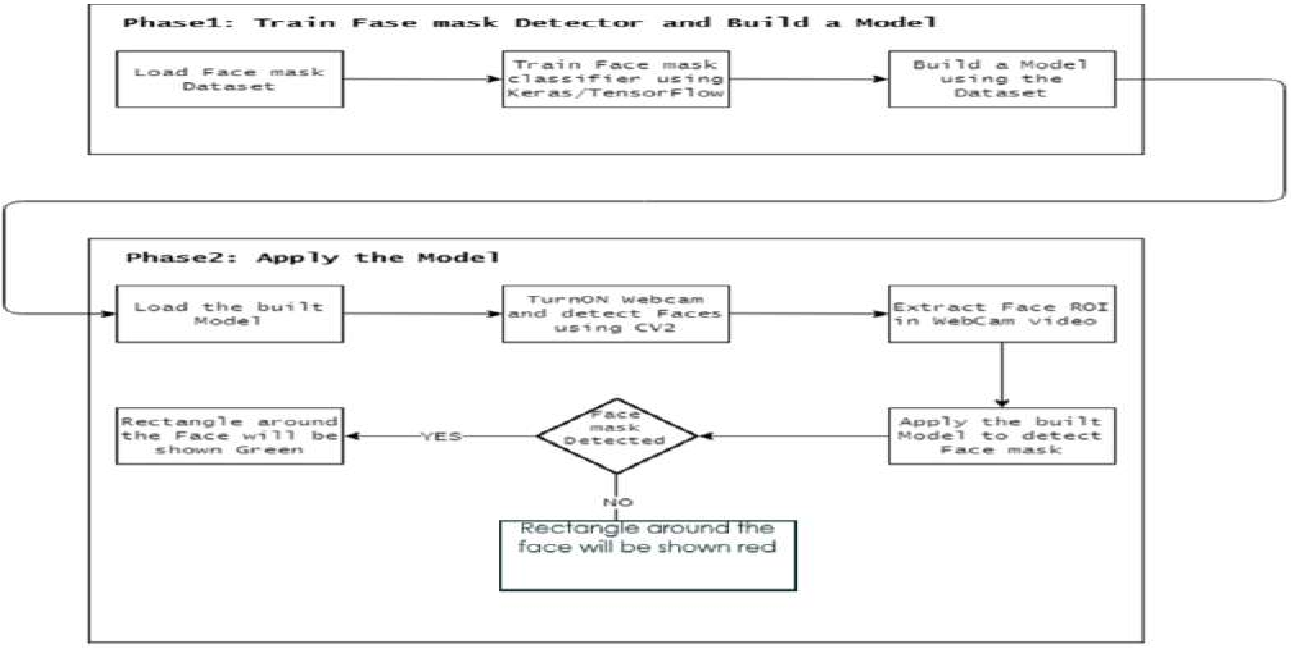
Combining an object detector with a dedicated with_mask class will allow improvement of the model in two respects.

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.

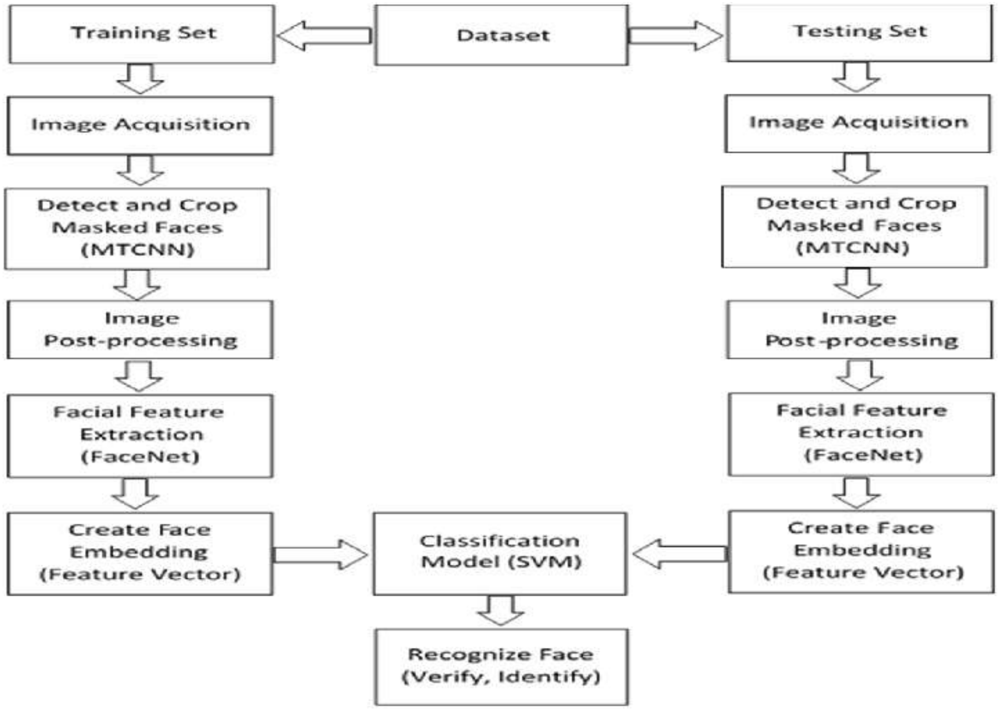
Secondly, this approach reduces our computer vision pipeline to a single step — rather than applying face detection and then our face mask detector model, all we need to do is apply the object detector to give us bounding boxes for people both with_mask and without_mask in a single forward pass of the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

Not only is such a method more computationally efficient, it's also more "elegant" and end-to-end.

UML Diagram:


Phase1: Train Fase mask Detector and Build a Model
Load Face mask Dataset
Train Face mask classifier using Keras/TensorFlow
Build a Model using the Dataset
Phase2: Apply the Model
Load the built Model
TurnON Webcam and detect Faces using CV2
Extract Face ROI in WebCam video
Apply the built Model to detect Face mask
Face mask Detected
YES
Rectangle around the Face will be shown Green
NO
Rectangle around the face will be shown red

UserFlow Diagram


Training Set
Dataset
Testing Set
Image Acquisition
Image Acquisition
Detect and Crop Masked Faces (MTCNN)
Detect and Crop Masked Faces (MTCNN)
Image Post-processing
Image Post-processing
Facial Feature Extraction (FaceNet)
Facial Feature Extraction (FaceNet)
Create Face Embedding (Feature Vector)
Classification Model (SVM)
Create Face Embedding (Feature Vector)
Recognize Face (Verify, Identify)

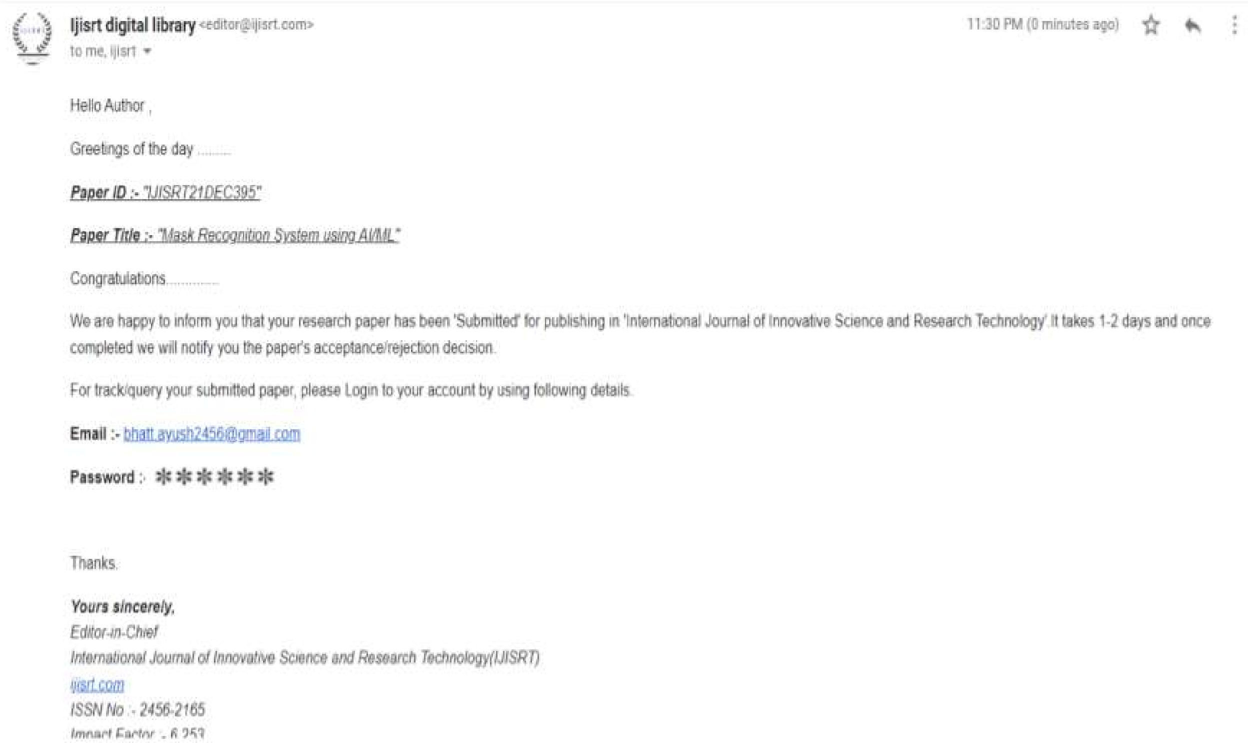
Future Scope

The model proposed is presently in phase I of operation and what we have shown is a direct application of it. However in future, following enhancements/ improvements can be superimposed to facilitate the ease of use and simplicity. These are described below:- Phase-I: Current version uses Laptop with relevant software installed As such, the requirement of such a loaded laptop with a camera is a must for execution of the software. Therefore the laptop should be installed at the point of inspection itself and some skilled operator should be present to type and execute the relevant commands. Phase-II: Replacement of laptop with a compact model with camera attached for video capture The plan is to replace the laptop with a workable model/ device such as Raspberry Pi which needs to be installed at the site of inspection/ site. Raspberry Pi models are compact in size. This will reduce installation complexities by 90%. This device/ model will be attached to a laptop only for execution of commands with help of LAN cable. As such commands can be run by a remote operator also and the requirement of laptop present at the site is negated. Phase-III: Removal of laptop for all purposes. Execution only with help of model There will not be any constraint of the laptop for executing any command to run the model either at the site or at any remote location. The device which is comparable to the size of a credit card with attached camera and loaded with necessary algorithms/code would meet the requirements. There would not be any obligation of trained/ semi trained individuals to run the commands. A normal sentry/ operator should be able to operate the model

CONCLUSION

Various methods and techniques for the detection and recognition of face masks are reviewed in this paper. In comparison, features like Haar are features of digital images used to visualize an object. They got their name because of their exact resemblance to the Haar waves and were used on a real-time face detector. The main advantage of a feature like Haar over many other features is its speed of calculation. Adaboost may be less prone to overloading than most learning algorithms. The worst feature of the flexible upgrade is its sensitivity to noisy data and external objects. In real world situations people may be covered by other things, such as face masks. This makes the facial recognition process a very challenging task. A method based on in-depth learning and quantitative-based approaches achieves high recognition performance. MobileNetV2 is a very efficient tool for object acquisition and segmentation. MobileNetV2 offers a highly efficient mobile-based model that can be used as the basis for many visual acuity tasks. To our best knowledge, this work deals with the problem of face recognition covered with masks and different methods during the COVID19 epidemic. We must point out that this research is not limited to this epidemic as many people are aware of it, take care of their health and wear masks to protect themselves from pollution and reduce the spread of germs.

Proof of Submission



Hello Author ,

Greetings of the day ....

Paper ID :- "IJISRT21DEC395"

Paper Title :- "Mask Recognition System using AI/ML"

Congratulations..............

We are happy to inform you that your research paper has been 'Submitted' for publishing in 'International Journal of Innovative Science and Research Technology'.It takes 1-2 days and once completed we will notify you the paper's acceptance/rejection decision.

For track/query your submitted paper, please Login to your account by using following details.

Email :- bhatt.ayush2456@gmail.com

Password :- ✻✻✻✻✻✻

Thanks.

Yours sincerely,
Editor-in-Chief
International Journal of Innovative Science and Research Technology(IJISRT)
ijisrt.com
ISSN No :- 2456-2165
Impact Factor :- 6.253

References:

1. Ariyanto, Mochammad & Haryanto, Ismoyo & Setiawan, Joga & Muna,Munadi & Radityo, M.. (2019). Real-Time Image Processing Method Using Raspberry Pi for a Car Model. 46-51.

2. V. K. Bhanse and M. D. Jaybhaye,(2018) "Face Detection and TrackingUsing Image Processing on Raspberry Pi," 2018 International Conferenceon Inventive Research in Computing Applications (ICIRCA),Coimbatore, India, pp. 1099-1103.

3. A. Das, M. Wasif Ansari and R. Basak, (2020) "Covid-19 Face MaskDetection Using TensorFlow, Keras and OpenCV," 2020 IEEE 17th IndiaCouncil International Conference (INDICON), New Delhi, India, pp. 1-5.

4. M. S. Islam, E. Haque Moon, M. A. Shaikat and M. Jahangir Alam,(2020) "A Novel Approach to Detect Face Mask using CNN," 2020 3rdInternational Conference on Intelligent Sustainable Systems (ICISS),Thoothukudi, India, pp. 800-806.

5. Joseph Redmon, S. D. (2016) “You Only Look Once(YOLO) Unified,Real Time Object Detection” IEEE.

6. A. Lodh, U. Saxena, A. khan, A. Motwani, L. Shakkeera and V. Y.Sharmasth, (2020) "Prototype for Integration of Face Mask Detection andPerson Identification Model – COVID-19," 2020 4th InternationalConference on Electronics, Communication and Aerospace Technology(ICECA), Coimbatore, India, pp. 1361-1367.

7. Luigi Atzori, Antonio iera and Giacomo Morabito. (2010) ‘The Internetof Things: A Survey’, Journal of Computer Networks Vol.54,No.15,pp.2787-2805.

8. Lu Tan and Neng Wang. (2010) ‘Future internet: The Internet of Things’,IEEE Xplore Proc., 3rd IEEE Int.Conf. Adv. Comp. Theory. Engg.(ICACTE), pp:1-9.

9. J. Marot and S. Bourennane, (2017)"Raspberry Pi for image processing education,"25th European Signal Processing Conference (EUSIPCO),Kos, Greece, 2017, pp. 2364-2366.40

10.S. A. Sanjaya and S. Adi Rakhmawan, (2020) "Face Mask DetectionUsing MobileNetV2 in The Era of COVID-19 Pandemic," 2020International Conference on Data Analytics for Business and Industry:Way Towards a Sustainable Economy (ICDABI), Sakheer, Bahrain, pp.1-5.

11.Senthilkumar.R and Gnanamurthy.R.K. (2016) ‘A Comparative Study of2D PCA Face Recognition Method with Other Statistically Based FaceRecognition Methods’, Journal of the Institution of Engineers India SeriesB (Springer Journal), Vol.97, pp.425-430.

12.Senthilkumar.R and Gnanamurthy.R.K. (2017) 'Performance Improvement in classification rate of appearance based statistical face recognition methods using SVM classifier”, the IEEE InternationalConference on Advanced Computing and Communication Systems(ICACCS), 6-7 January 2017, pp. 286-292. 46

13. Senthilkumar.R and Gnanamurthy.R.K. (2018) ‘HANFIS: A New Fast And Robust Approach for Face Recognition and Facial ImageClassification’, Advances in Intelligent Systems and Computing SmartInnovations in Communication and Computational Sciences, Chapter 8,pp:81-99.

14.S. Susanto, F. A. Putra, R. Analia and I. K. L. N. Suci Ningtyas, (2020)"The Face Mask Detection For Preventing the Spread of COVID-19 atPoliteknik Negeri Batam," 2020 3rd International Conference on AppliedEngineering (ICAE), Batam, Indonesia, pp. 1-5.

15. S. S. Walam, S. P. Teli, B. S. Thakur, R. R. Nevarekar and S. M. Patil,(2018) "Object Detection and Separation Using Raspberry PI," 2018Second International Conference on Inventive Communication andComputational Technologies (ICICCT), Coimbatore, India, pp. 214-217.

16. Yair Meidan, Michael Bohadana, Asaf Shabatai, Juan David Guarnizoand NilsOle Tippenhauer and Yuval Elovici. (2017) ‘ProfilloT: a machine learning approach for IoT device identification of network traffic analysis’, proc.Sym. App. Comp., SAC, pp: 506-509.

17.G. Yang et al.,(2020) "Face Mask Recognition System with YOLO V5 Based on Image Recognition," 2020 IEEE 6th International Conferenceon Computer and Communications (ICCC), Chengdu, China, pp.1398-1404.