Real time Face Mask Detection System

Saumy Srivastava Follow
Jul 21, 2020 · 6 min read

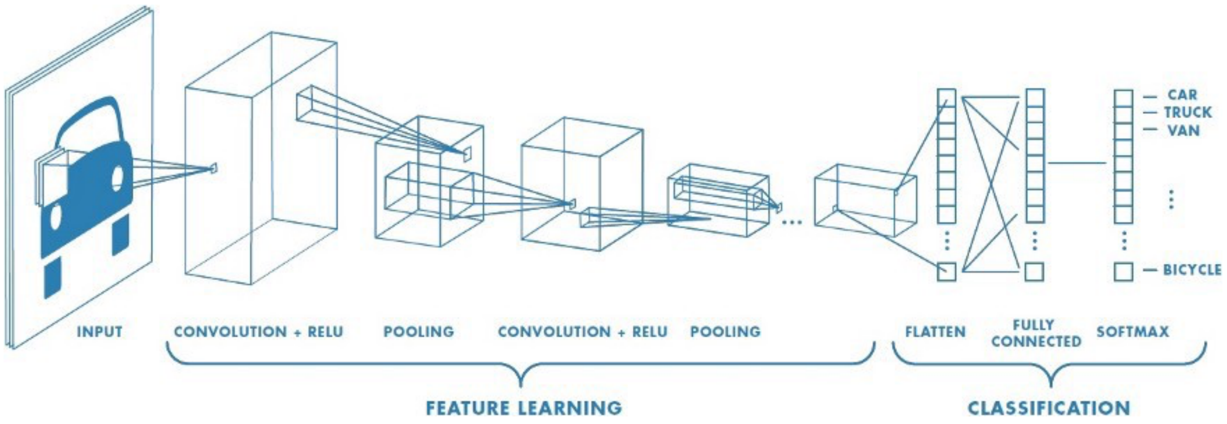
Real Time Face Mask Detector

> *"Productivity is being able to do things that you were never able to do before."* — Franz Kafka

The recent coronavirus pandemic has pushed humans all around the world to the new challenges. In this context of uncertainty, we can all play our role by contributing to the fight against this disease. This is an excellent opportunity to put technology at the service of humanity. So, we can contribute towards this epidemic by implementing the productive things in its favour.

What if we had a system that could monitor whether people around us are complying with all the safety measures? So why not to build a very simple and basic Convolutional Neural Network (CNN) model using ***TensorFlow*** with ***Keras*** library and ***OpenCV*** project to detect if you are wearing a face mask to protect yourself, which also shows you accuracy of it.

Convolutional Neural Network (CNN) — Deep Learning



Neural network with many convolutional layers

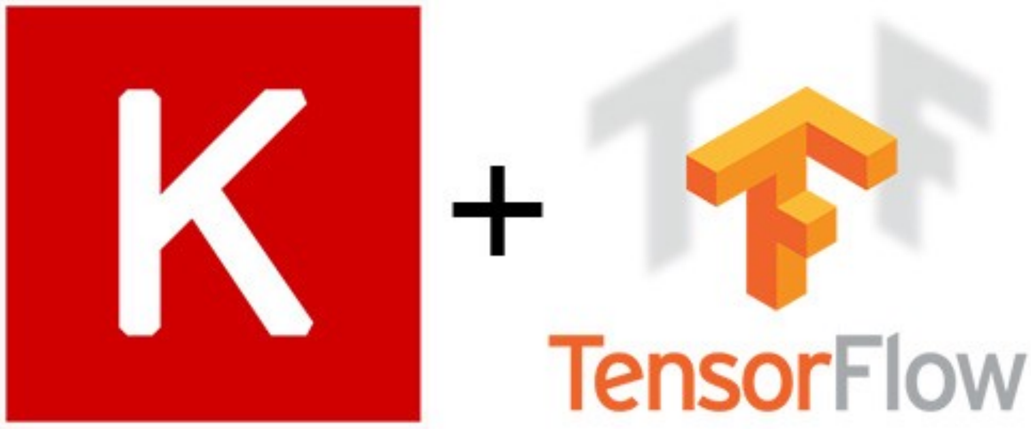
CNN's are inspired by the structure of the brain but our focus will not be on neural science here as we do not have any expertise or academic knowledge in any of the biological aspects. Also to note they are nowhere close to the actual human brain. CNN's are a class of Neural Networks that have proven very effective in areas of image recognition, processing, and classification.

As per Wiki — *In* ***machine learning****, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks, most commonly applied to analysing visual imagery.*

They exist already for several decades but were shown to be very powerful when large labeled datasets are used. This requires fast computers (e.g. GPUs)!

This network is a great example of variation for multilayer perceptron for processing and classification. It’s a deep learning algorithm in which it takes input as an image and put weights and biases effectively to its objects and finally able to differentiate images from each other.

TensorFlow with Keras Library



TensorFlow with Keras

In this , we’ll be talking about two of the many libraries and frameworks — TensorFlow and Keras.

What is TensorFlow?

TensorFlow is an open-sourced library. It is one of the most famous libraries when it comes to dealing with Deep Neural Networks. The primary reason behind the popularity of TensorFlow is the sheer ease of building and deploying applications using TensorFlow.

TensorFlow

TensorFlow excels at numerical computing, which is critical for deep learning. It provides APIs in most major languages and environments needed for deep learning

What is Keras?

Keras is a completely Python-based framework, which makes it easy to debug and explore. It is a high-level library that's built on top of TensorFlow. It provides a scikit-learn type API for building Neural Networks. Developers can use Keras to quickly build neural networks without worrying about the mathematical aspects of tensor algebra, numerical techniques, and optimization methods.

The key idea behind the development of Keras is to facilitate experiments by fast prototyping. The ability to go from an idea to result with the least possible delay is key to good research.

This offers a huge advantage for scientists and beginner developers alike because they can dive right into Deep Learning without getting their hands dirty with low-level computations.

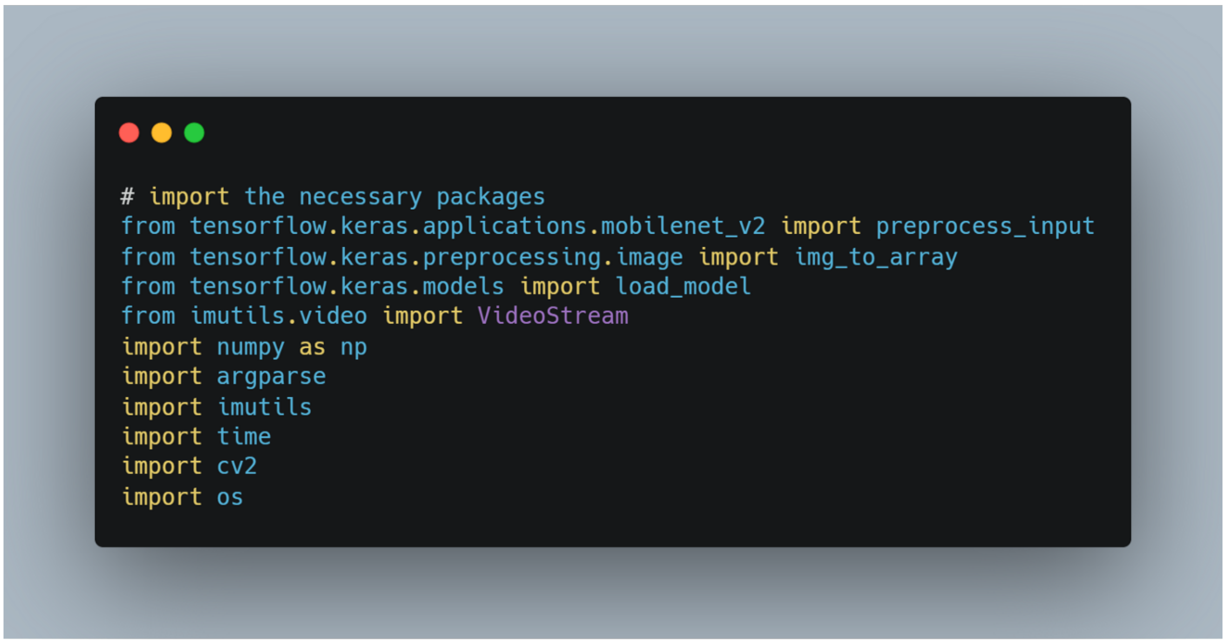
OpenCV

OpenCV is a cross-platform library using which we can develop real-time **computer vision applications**. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

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

Let see how to detect face with real time video stream. First install necessary packages.

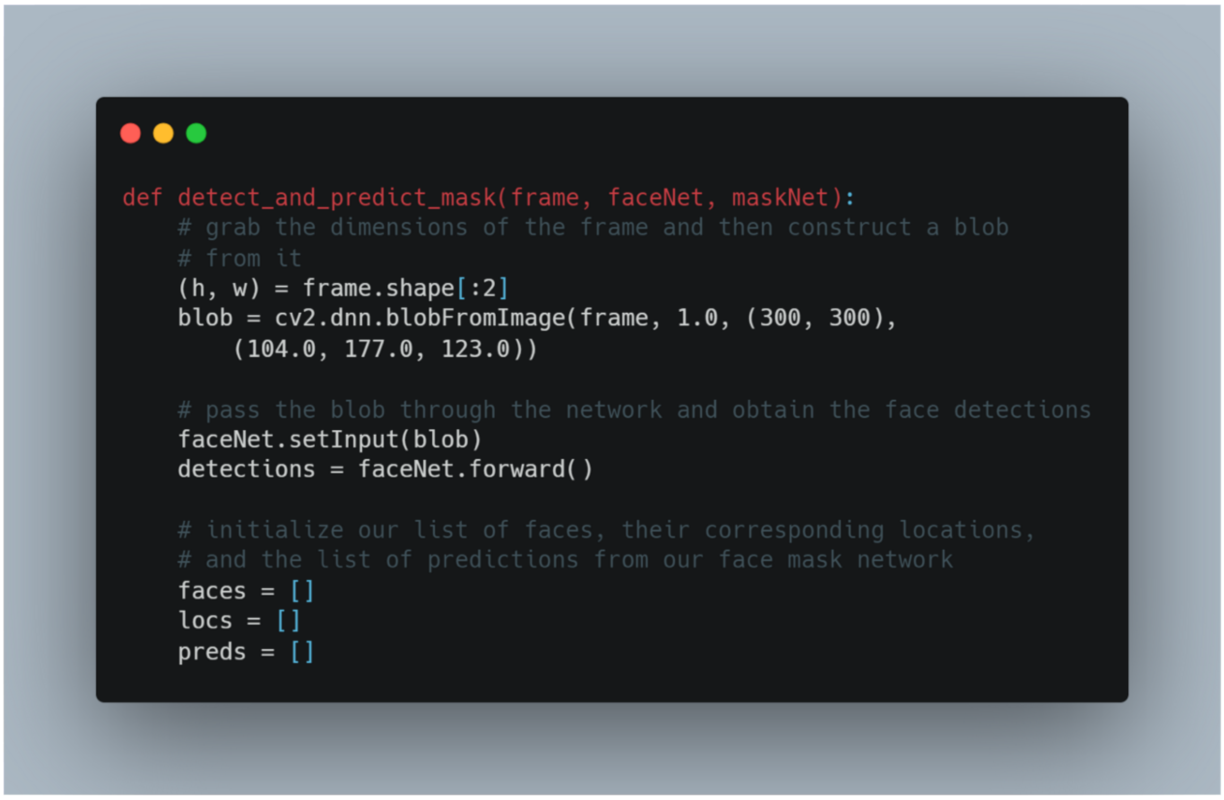


```python
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import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

Importing Necessary Packages

Function which detects the faces and then applies our face

mask classifier to each face.



```python
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 = []
```

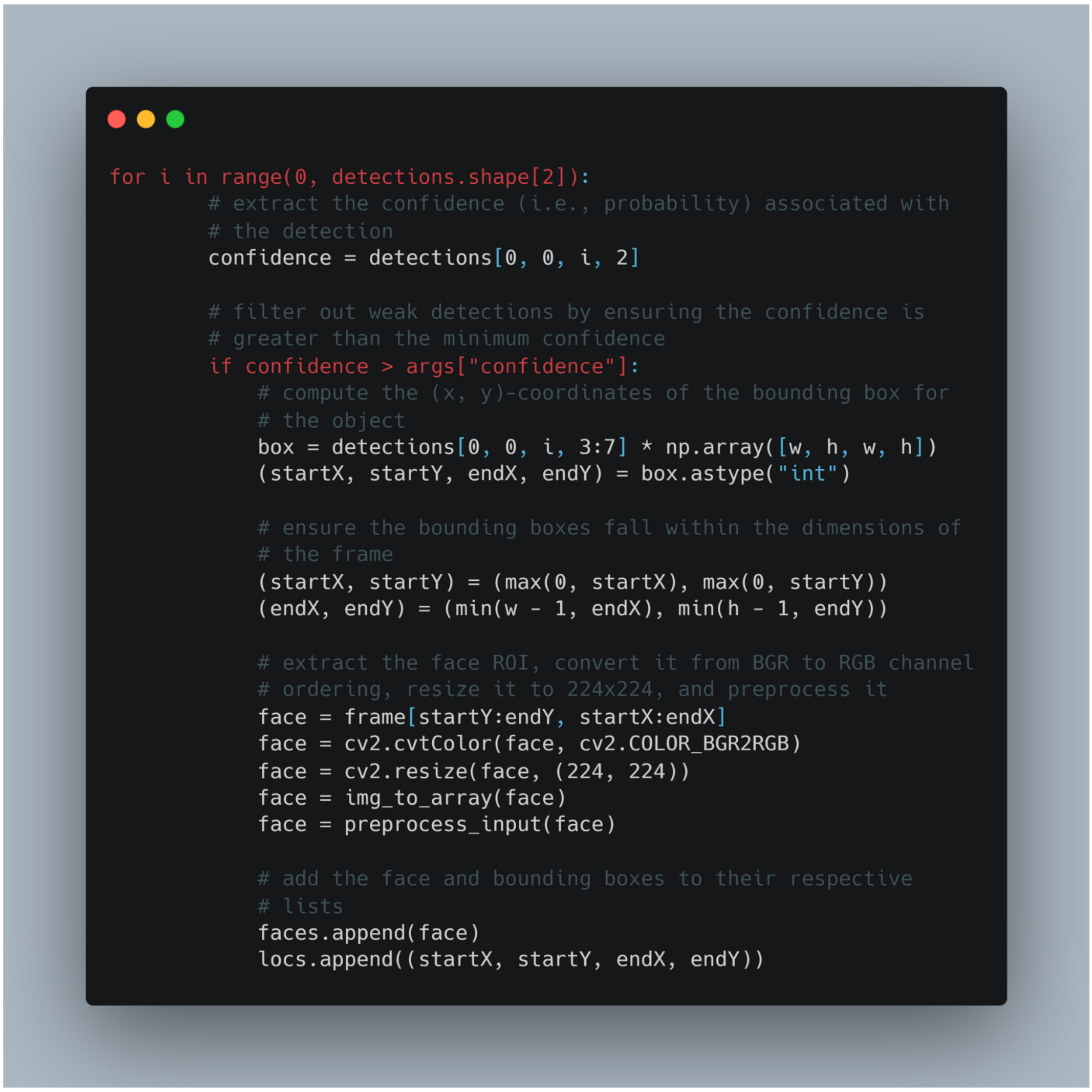
Function to detect faces and appling mask classifier

The above mentioned detect_and_predict_mask function accepts three parameters:

- frame: A frame from our stream
- faceNet : This is the model used to detect the faces inside the image
- maskNet : The 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).

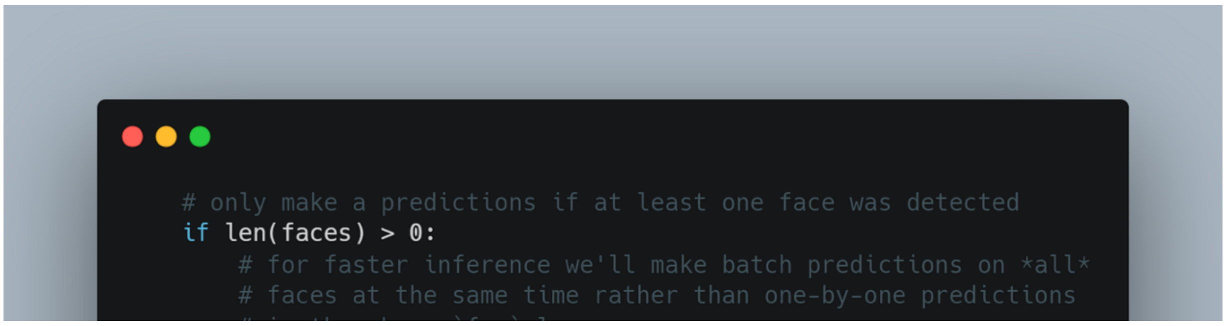
From here, we'll loop over the detections of faces:



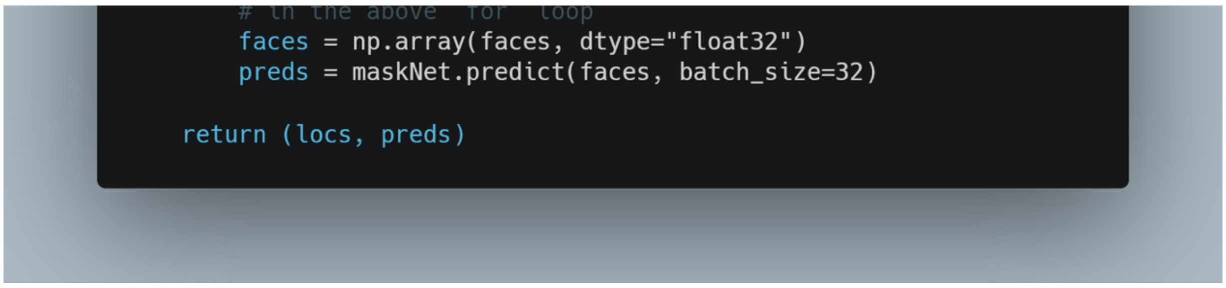
```
for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

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

            # 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))
```

Loop over face detection

Now, we can implement the mask predictor created for predicting the faces.

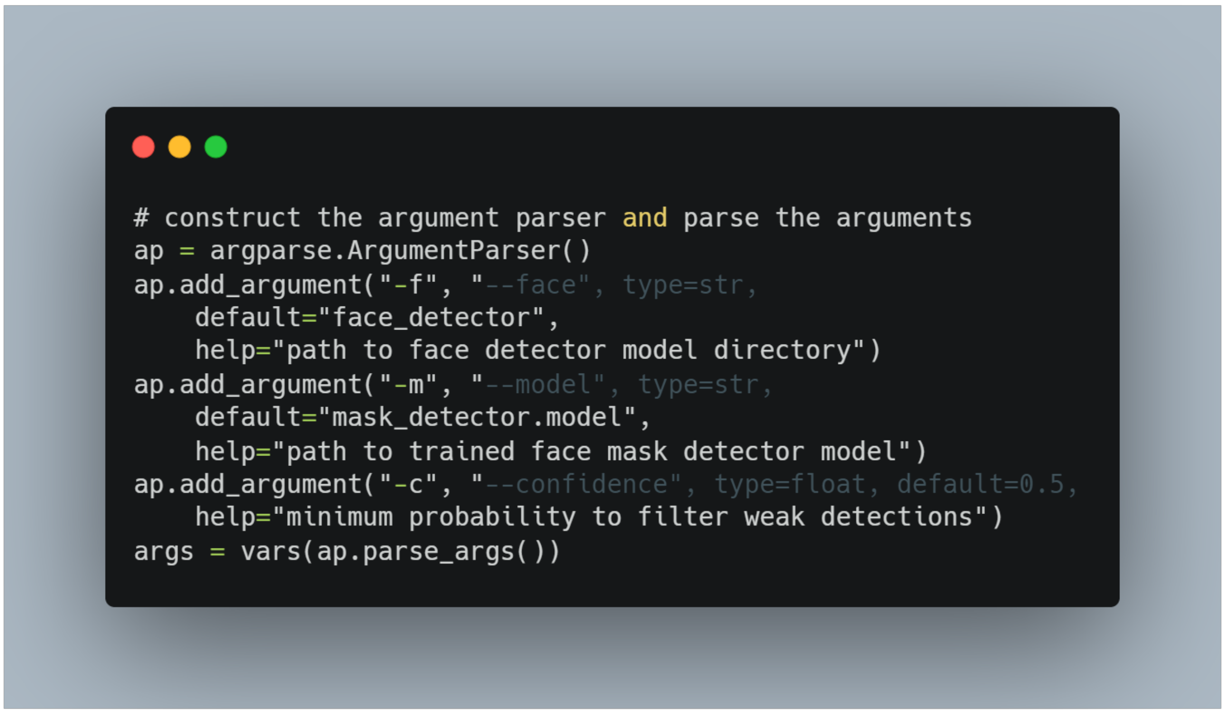


```
    # only make a predictions if at least one face was detected
    if len(faces) > 0:
        # for faster inference we'll make batch predictions on *all*
        # faces at the same time rather than one-by-one predictions
```



```
        # in the above 'for' loop
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    return (locs, preds)
```

Next, defining some of the command line arguments:

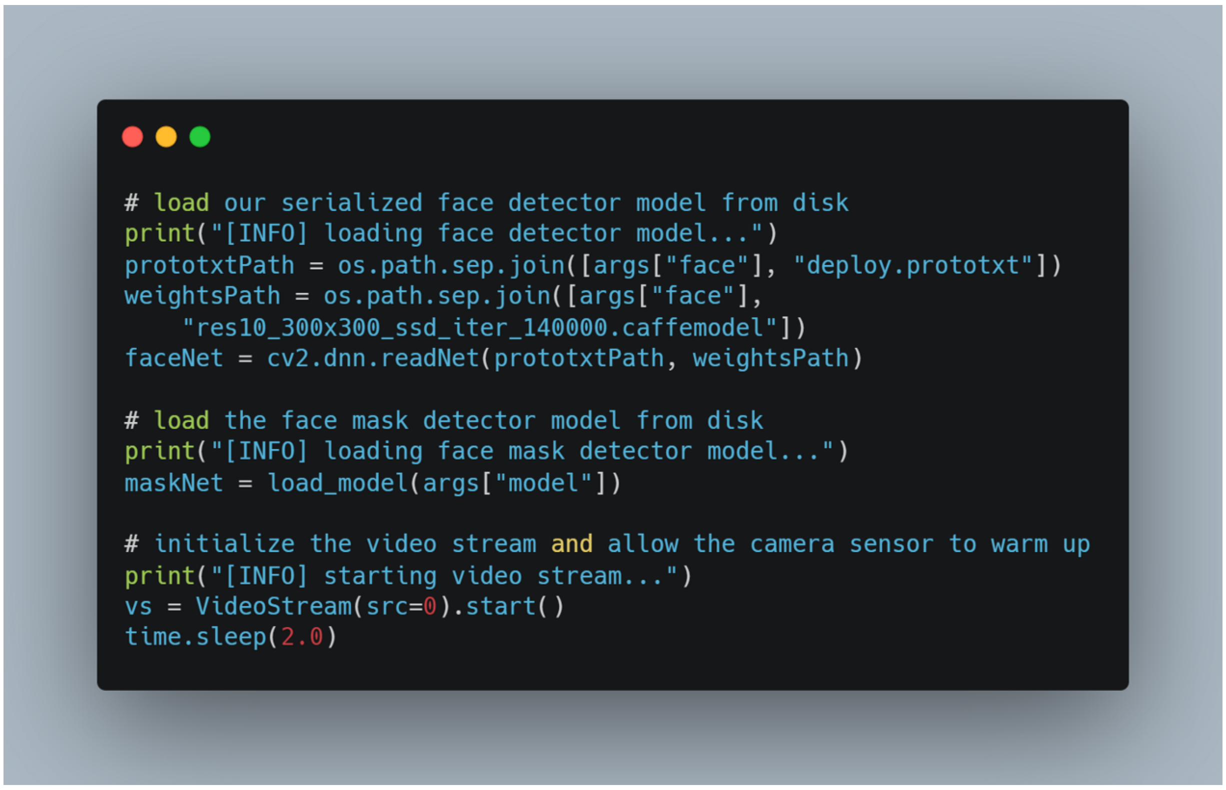


```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path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

The command line arguments mentioned above includes:

- 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:

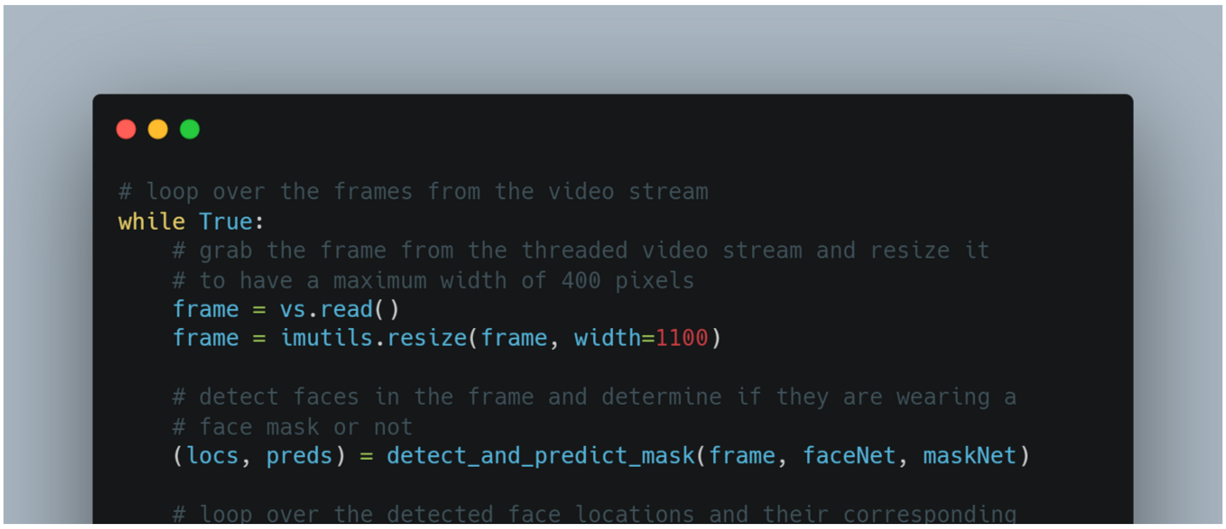


```python
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

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

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

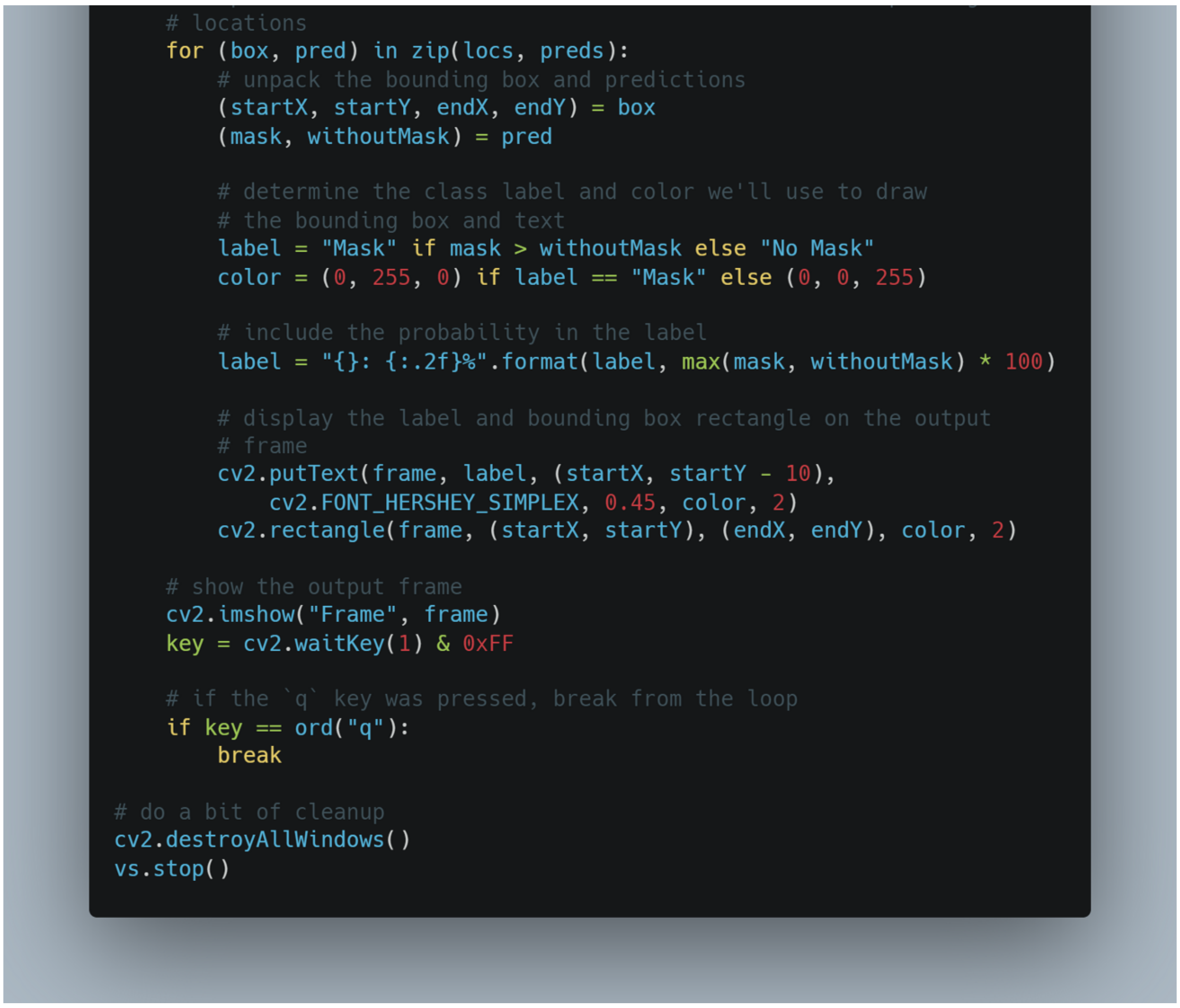
Here we have initialized :

- Face detector
- face mask detector
- Webcam video stream

Now, we'll proceed to loop over frames from the live webcam video stream:



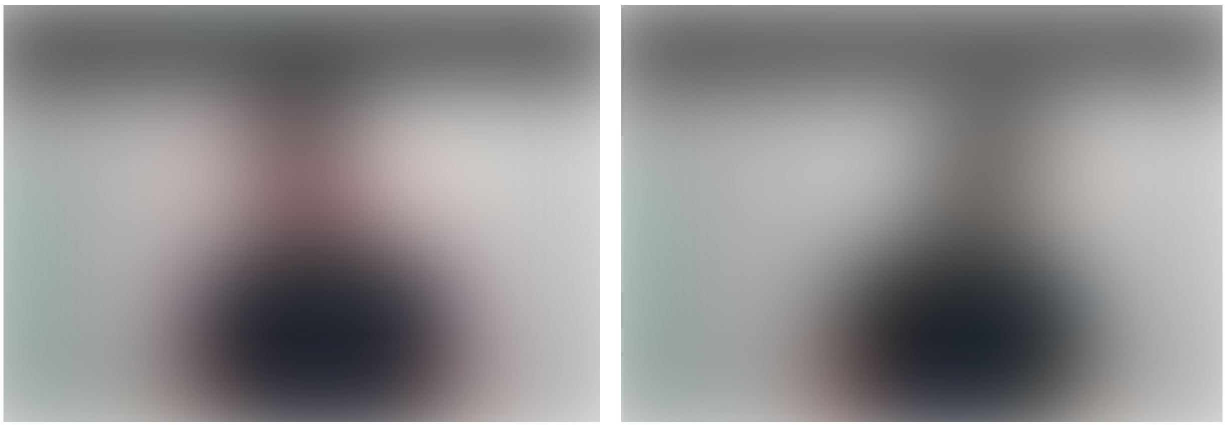
```python
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=1100)

    # 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()
```

Loop over frames from the stream

All the above mentioned code snippets are for detecting the face mask in live webcam video stream. It is just for understanding the Detection through live stream part. Thus, implementation of real-time face mask detector with Python, OpenCV, and deep learning with TensorFlow/Keras can be done.


Face Mask Detection

From above snapshots, we can see it was correctly able to detect if I was wearing a mask or not and displays the mask accuracy and that was also pretty good.

In this article, we saw how to detect if a person is wearing a face mask or not. This can be used in numerous applications. Wearing a mask may be necessary in the near future, considering the present epidemic and this method to detect if the person wears a face mask may come very useful.

Stay Safe!

TensorFlow Keras Opencv Covid 19 Face Mask Detection