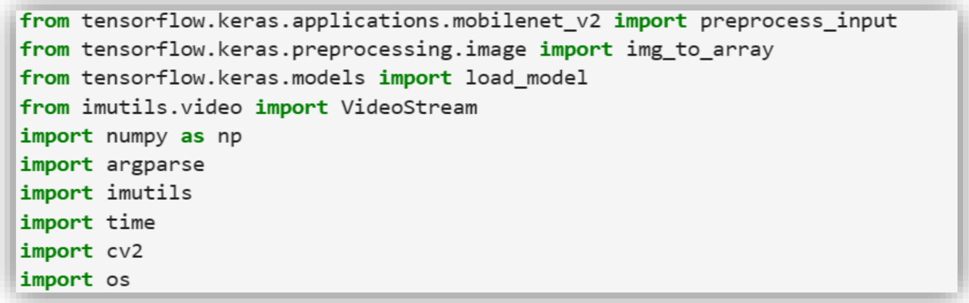
Face Mask Detection Application flow

1. **Check if the following mandatory files available before execute of the script.**

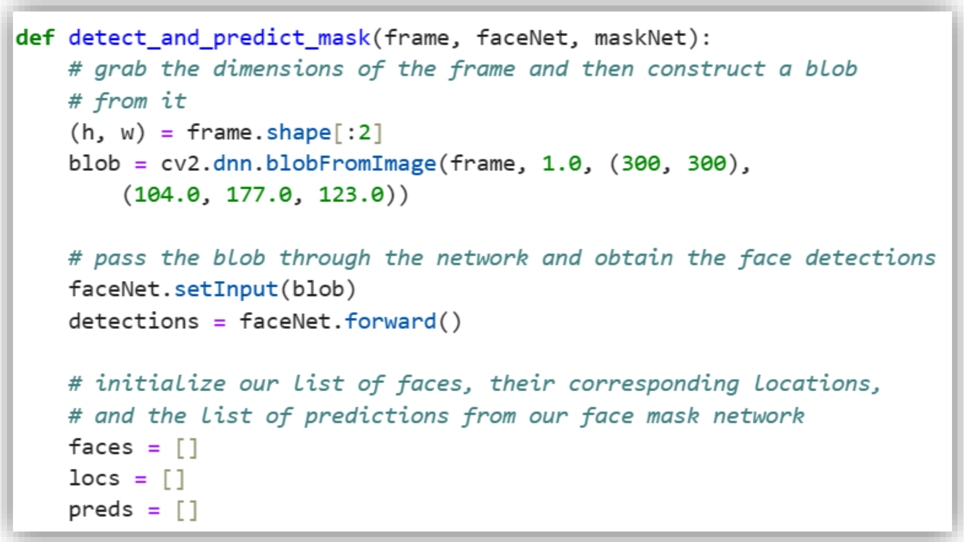
- ✓ Dataset (contains sub-folder with two classes “with_Mask” and “without_Mask” contains images
- ✓ Training program file
- ✓ Best model save or deployment file
- ✓ Application file for execution of the final output.

2. **Load the necessary libraries and below is snippet of those.**



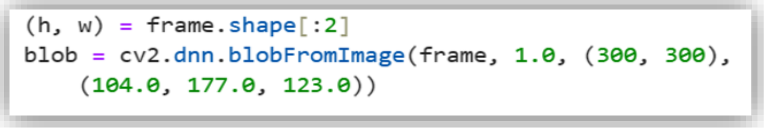
```python
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2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

3. **Define a function to be able to input frame, facenet, masknet**
   - **frame :** To take an image or series image in a video stream
   - **facenet :** To deduct a region where face located in the image
   - **masknet :** To deduct if the face deducted has mask or no mask



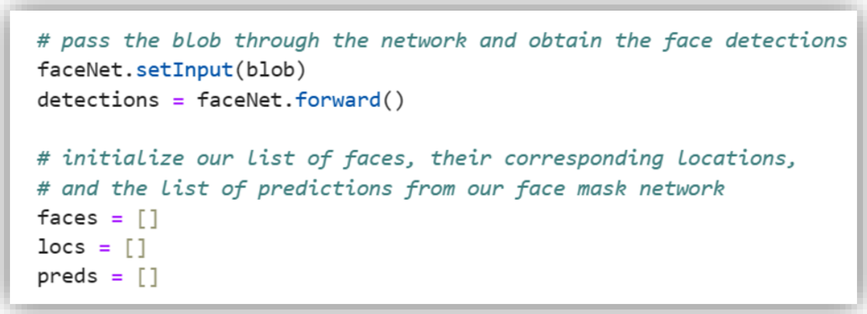
```python
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
        (104.0, 177.0, 123.0))

    # 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 = []
```

Get the “Height” and “Width” of the frame (image taken) and exclude channels (RGB) from it and then using computer vision lib. Deep neural networks covert the frame into blob (binary large object) and assign it in a variable called blob. During the course of assignment taking frame original lighting effect (scaling = 1.0), set the pixel size (300,300) and set 3 color channel which basically mean subtraction and scaling compatible to be able to input in any pre-trained model (which is already created by experts)



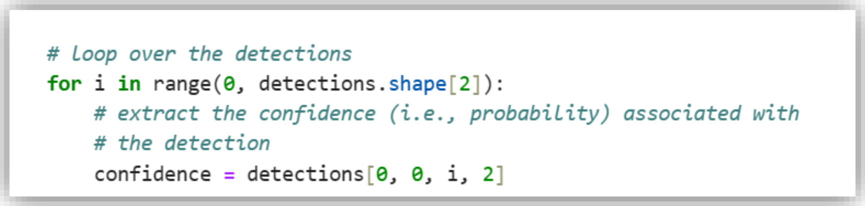
```python
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
    (104.0, 177.0, 123.0))
```

FaceNet is a pre-trained neural network model able to deduct faces appropriately using “setInput” we are inputting the blob and detect the fact using “faceNet.forward()” and assign that in a variable called detections. Alongside creating an empty list to be able to append list of faces, their locations and prediction if those faces have mask or no mask.



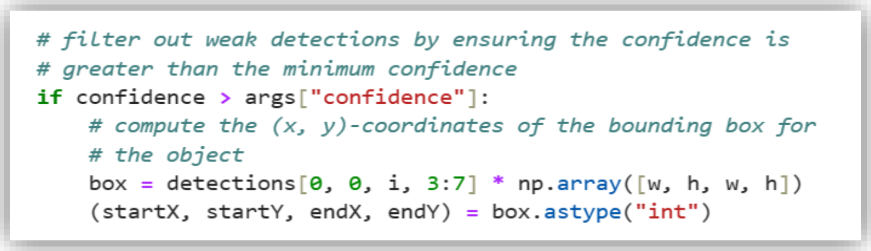
```
# pass the blob through the network and obtain the face detections
faceNet.setInput(blob)
detections = faceNet.forward()

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

Using for loop iterating from ‘0’ (from Height) and exactly extract number of faces (detections.shape[2]) and determine the confidence score i.e., the deducted faces, closely observe its part that stands for detections[0,0], ‘i’ stands selected/iterate face and 2 stands for confidence score level the default minimum score begins at 0.5 and this being assigned to a variable called confidence.



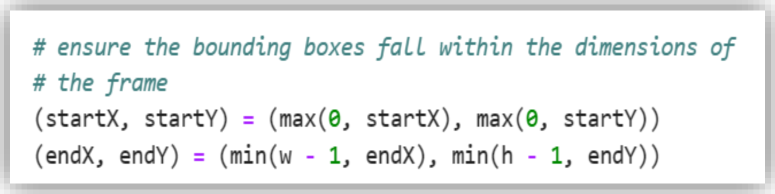
```
# loop over the detections
for i in range(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with
    # the detection
    confidence = detections[0, 0, i, 2]
```

Using if statement checking condition if confidence score greater than 50%, if so deducted faces, closely observe its part that stands for detections[0,0], ‘i’ stands selected/iterate face and form left, top, right and bottom later [3:7] and that multiply with np.array to covert to width (Top layer), height(Left layer), width (Bottom layer), height(Right layer) and assign them to a variable called box. width (Top layer), height(Left layer), width (Bottom layer), height(Right layer) assigned them to an individual variable called startX, startY,endX,endY and convert np.array float values into int to get a bounding box.



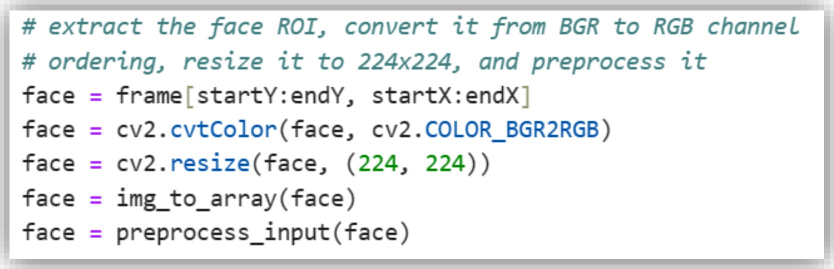
```
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > args["confidence"]:
    # compute the (x, y)-coordinates of the bounding box for
    # the object
    box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
    (startX, startY, endX,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

To make sure that the bounding box fall within the dimension “(startX, startY) = (max(0, startX), max(0, startY))” forms top and left line that doesn’t go beyond its dimension specified in the left width and the top height of StartX , StartY. Similarly, (endX, endY) = (min(w - 1, endX), min(h - 1, endY)) forms Bottom and Right line that doesn’t go beyond its dimension specified in the right width and the bottom height of endX , endY.



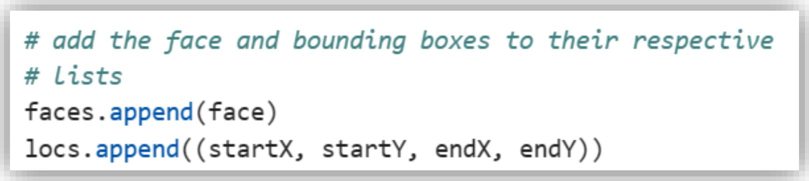
```
# ensure the bounding boxes fall within the dimensions of
# the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
```

- ✓ Taking frame heights of left and right, width of top and bottom respectively and assign to face variable.
- ✓ Covert the face color from BGR and RBG using computer vision lib because the pre-trained model can learn train using different color palate of the frame and it is usual while we call up any pre-trained model and assign it to variable face.
- ✓ Resize the image into 224 x 224 to be able to accept by the pre-trained model and assign it to variable face.
- ✓ Convert the face into array using img_to_array and assign it to variable face.
- ✓ Finally preprocess the input face to be able to handle by pre-trained model and assign it to variable face.



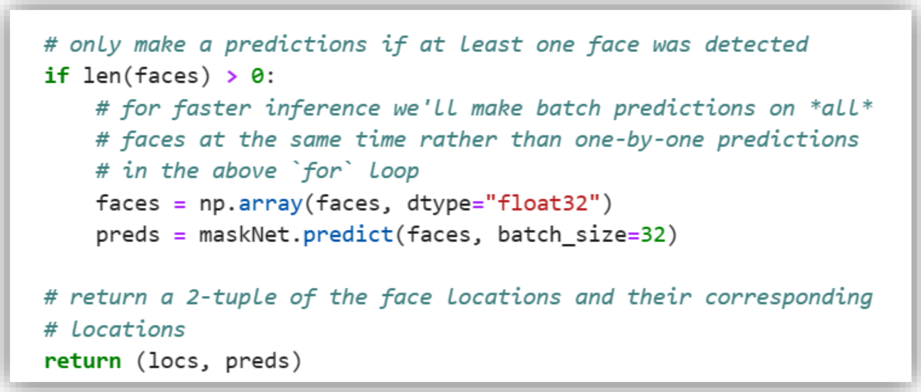
```
# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
face = frame[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
```

The preprocess face appended into the empty face list created and the empty locs list appended with variables of the bounding box i.e., startX, startY, endX, endY.



```
# add the face and bounding boxes to their respective
# lists
faces.append(face)
locs.append((startX, startY, endX, endY))
```

Using if statement, instruction to make prediction if at least 1 face detected and convert the input face into np.array to be able to understand by neural networks model and make it data type as float to improve the prediction more appropriate and assign to variable face. Using pre-trained model MaskNet predicted the given face has mask or no mask likewise the cam able to deducted 32 faces through a batch size of 32 and finally return it to locs (location of the face) and preds (where stored pre-trained mask defection model)

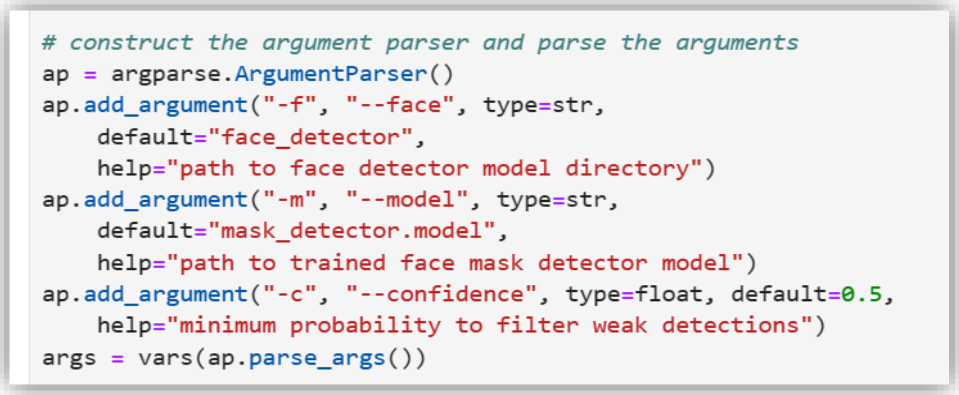


```
# only make a predictions if at least one face was detected
if len(faces) > 0:
    # for faster inference we'll make batch predictions on *all*
    # faces at the same time rather than one-by-one predictions
    # in the above `for` loop
    faces = np.array(faces, dtype="float32")
    preds = maskNet.predict(faces, batch_size=32)

# return a 2-tuple of the face locations and their corresponding
# locations
return (locs, preds)
```

Here we are using argument parser for the script to be written in command-line console

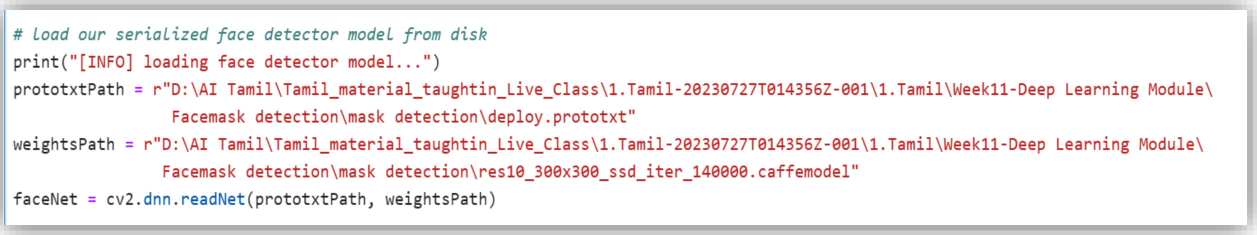
In the argument parser (ap) adding the argument where the system can access “face_detector” folder (which has config. Files “deploy.prototxt”, “mask_detector.model”, “res10_300x300_ssd_iter_140000”) will be access at the time of running the script while running the program

Also declaring the minimum confidence score which is called at time of checking the condition to filter confidence score greater than the min of 50%



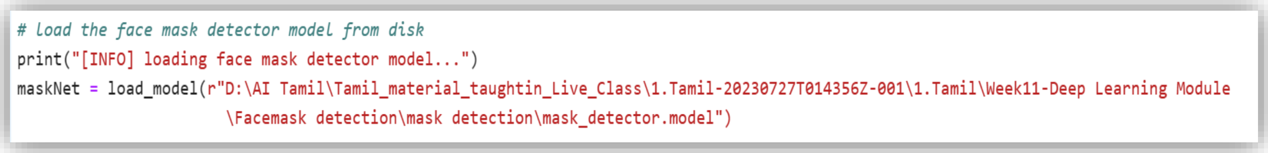
```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path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

Mentioning the path to load deploy config file in the variable Prototxtpath as raw string
Mentioning the path of weight in the config file in the variable weightspath as raw string
Finally, computer vision lib. Deep neural network. readNet method reads both deployment and weight config file and assigned to the variable faceNet to detect face in a frame.



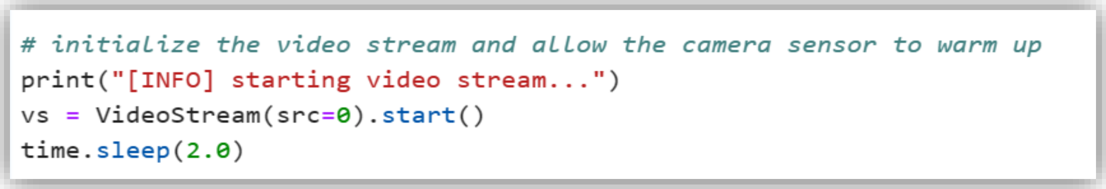
```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = r"D:\AI Tamil\Tamil_material_taughtin_Live_Class\1.Tamil-20230727T014356Z-001\1.Tamil\Week11-Deep Learning Module\
                Facemask detection\mask detection\deploy.prototxt"
weightsPath = r"D:\AI Tamil\Tamil_material_taughtin_Live_Class\1.Tamil-20230727T014356Z-001\1.Tamil\Week11-Deep Learning Module\
               Facemask detection\mask detection\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

Mentioning the path to load face mask detector model as raw string using load_model method and assign it to a variable MaskNet.



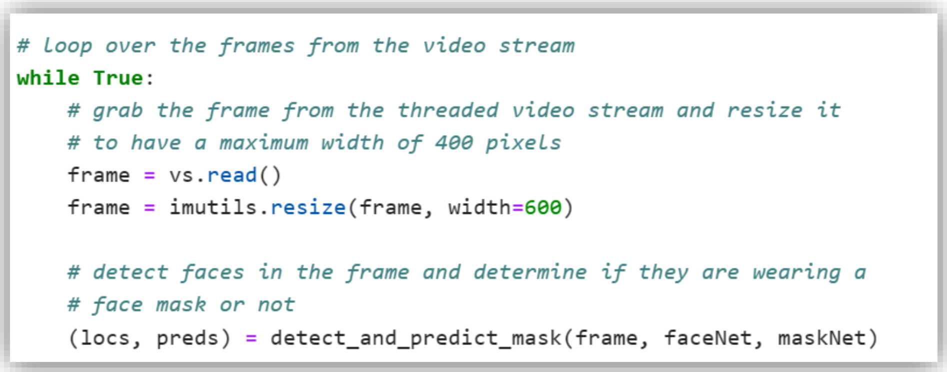
```
# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(r"D:\AI Tamil\Tamil_material_taughtin_Live_Class\1.Tamil-20230727T014356Z-001\1.Tamil\Week11-Deep Learning Module
                     \Facemask detection\mask detection\mask_detector.model")
```

Using videoStream method, start stream using source = our local lap webcam (src = 0) and this action will be assigned to variable vs and make 2 milliseconds to delay within which webcam can adjust to capture internally.



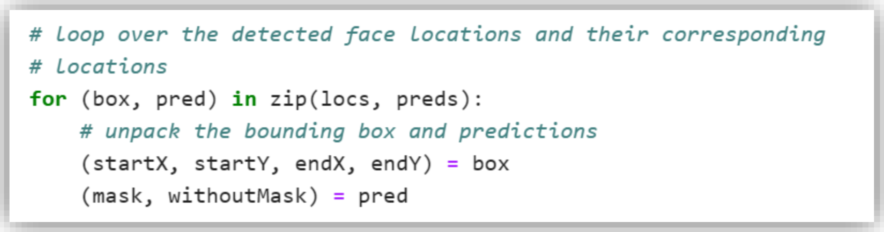
```
# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

Using while statement (while set to True) continuously looping through the frames of the video stream and captured vs will be assigned to a variable frame and then from imutils lib resize the frame of vs. finally function detect_and_predict_mask assigned to variable locs , preds (locs refers to location of the face and predicting if the face has mask or no mask from the frame)



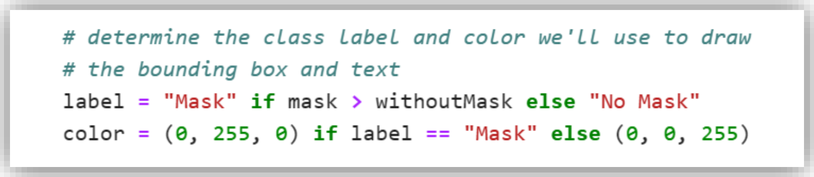
```python
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=600)

    # 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)
```

Using for loop iterating the unpacked locs, preds via box, pred temporary variables. locs being iterated by box and unpacked startX(top), startY(left),endX(bottom),endY(right). Preds being interated by pred and unpacked mask, withoutmask.



```python
# 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
```

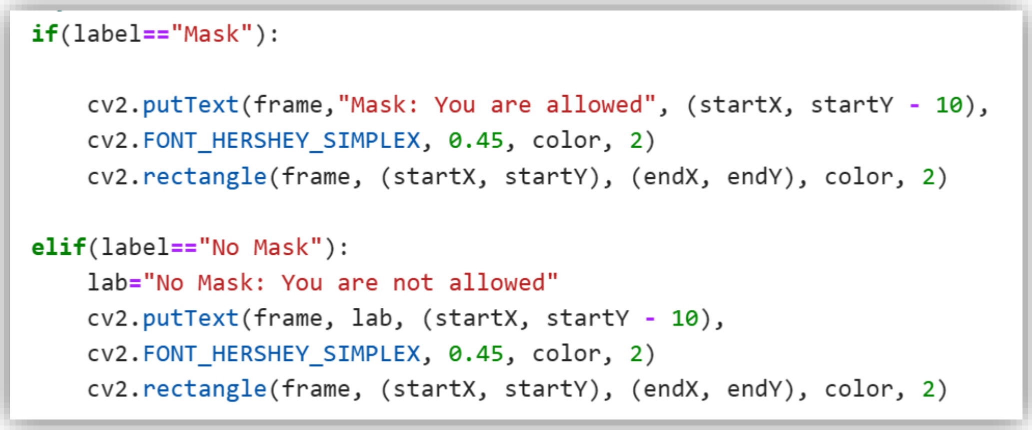
In training part of program one hot encoding used which indicates Mask (1), withoutMask (0). Hence the condition is if 1 (mask) > 0 (withoutmask) label it as “Mask” and color it Green (RGB 0, 255, 0) otherwise (0 (withoutmask) > 1 (mask) label it as “No Mask” and color it RED (RGB 0,0,255)



```python
    # 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)
```

If label is Mask, using computer vision. putText method in the frame show “Mask: You are allowed" (startX, startY - 10) is to a position i.e, just above the top layer place the text, cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2 is the font style being offered by cv2 and its size is of 45%, color is Green (which already specific in the color variable using if condition above), 2 is the thickness of the bounding box line.

Finally using computer vision bounding box formed using rectangle method color it based on the condition mentioned above and the box thickness is of.



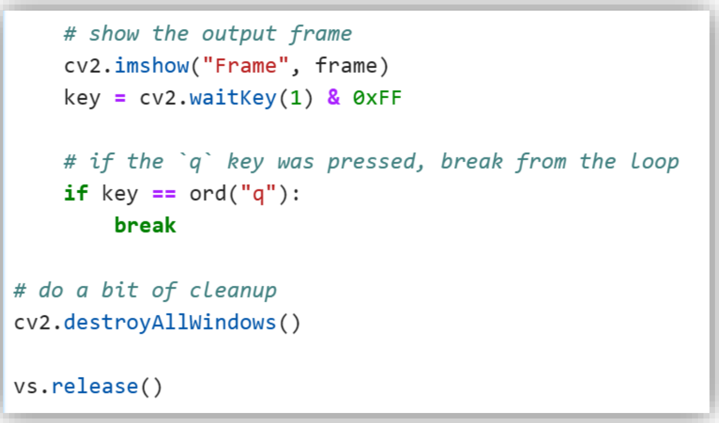
```
if(label=="Mask"):

    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)
```

elif label is No Mask, using computer vision. putText method in the frame show “No Mask: You are not allowed" (startX, startY - 10) is to a position i.e, just above the top layer place the text, cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2 is the font style being offered by cv2 and its size is of 45%, color is Red (which already specific in the color variable using if condition above), 2 is the thickness of the bounding box line.

Finally using computer vision bounding box formed using rectangle method color it based on the condition mentioned above and the box thickness is of.

Using computer vision show image (using imshow method) and take input as frame and the processed frame titled Frame.

waitKey method waits for a key press event for 1 millisecond. The result of this function is assigned to the variable key. The & 0xFF operation is often used to extract the ASCII value of the key. Since the mask detection continuously executes to terminate it need to hit “q” from the keyboard so the if condition check if the key press == ‘q’ that camera window closes and released from video streaming



```
    # 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.release()
```