Code Explanation and Documentation

The following code is a Python script that uses a pre-trained deep learning model to detect whether a person is wearing a face mask or not in a real-time video stream.

Importing the necessary packages

‘’’

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.models import load_model

from imutils.video import VideoStream

import numpy as np

import imutils

import time

import cv2

import os

‘’’

- `tensorflow.keras.applications.mobilenet_v2`: Importing the MobileNetV2 model, which is used for face mask detection.
- `tensorflow.keras.preprocessing.image`: Importing functions for preprocessing images.
- ‘tensorflow.keras.models’: Importing the function to load a pre-trained model.
- `imutils.video.VideoStream`: Importing the `VideoStream` class for capturing frames from a video stream.
- `numpy`: Importing the NumPy library for numerical computations.
- `imutils`: Importing utility functions for resizing and manipulating images.
- `time`: Importing the `time` module for time-related operations.
- `cv2`: Importing the OpenCV library for computer vision tasks.
- `os`: Importing the `os` module for interacting with the operating system.

Defining the `detect_and_predict_mask` function

This function takes an input frame, a face detection network (`faceNet`), and a face mask detection network (`maskNet`). It detects faces in the given frame and predicts whether the detected faces are wearing masks or not.

‘’’

def detect_and_predict_mask(frame, faceNet, maskNet):

    # grab the dimensions of the frame and then construct a blob

    # from it

    (h, w) = frame.shape[:2]

    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),(104.0, 177.0, 123.0))

‘’’

- `frame`: The input frame to be processed.
- `faceNet`: The pre-trained face detection network.
- `maskNet`: The pre-trained face mask detection network.

The function starts by grabbing the dimensions of the frame and creating a blob from it. A blob is a format of image representation used as input by deep learning models.

‘’’

    # pass the blob through the network and obtain the face detections

    faceNet.setInput(blob)

    detections = faceNet.forward()

    print(detections.shape)

‘’’

The blob is passed through the face detection network (`faceNet`) to obtain face detections. The network predicts the locations of faces in the image. The resulting detections are printed to the console for debugging purposes.

```

    # initialize our list of faces, their corresponding locations,

    # and the list of predictions from our face mask network

    faces = []

    locs = []

    preds = []
```

```

Lists are initialized to store the detected faces, their corresponding locations, and the predictions from the face mask detection network.

```

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

        # filter out weak detections by ensuring the confidence is
        # greater than the minimum confidence
        if confidence > 0.5:
            # compute the (x, y)-coordinates of the bounding box for
            # the object
            box = detections[0, 0, i, 3:7

] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

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

The code iterates over the face detections and processes each detected face. It extracts the confidence associated with each detection and filters out weak detections by applying a minimum confidence threshold (0.5).

For valid detections, it computes the bounding box coordinates (`box`) relative to the original frame size and ensures that the bounding box coordinates fall within the frame dimensions.
```

```
        # 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))
```

For each valid face detection, the face region of interest (ROI) is extracted from the frame. The ROI is converted from BGR to RGB color ordering, resized to 224x224 pixels (required input size for the face mask detection network), and preprocessed using the `img_to_array` and `preprocess_input` functions.

The preprocessed face and its bounding box coordinates are added to their respective lists (`faces` and `locs`).

```
    # 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)
```

If at least one face is detected, the code converts the `faces` list to a NumPy array and performs batch predictions on all the faces at once using the face mask detection network (`maskNet`). This batch prediction is more efficient than making predictions one by one in the previous loop.

The predictions are stored in the `preds` list.

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

Finally, the function returns a tuple containing the face locations (`locs`) and their corresponding predictions (`preds`).

Loading the pre-trained models and starting the video stream

```python
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

maskNet = load_model("mask_detector.model")

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
```

The code loads the pre-trained face detection model (`faceNet`) using the provided `.prototxt` and `.caffemodel` files. It also loads the pre-trained face mask detection model (`maskNet`) using the `load_model` function. The paths to the models

are provided as strings.

A video stream is started using the `VideoStream` class, and the frames are captured from the default video source (webcam).

Processing frames and performing mask detection

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

    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        # include the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        # display the label and bounding box rectangle on the output
        # frame
        cv2.putText(frame, label, (startX, startY - 10),
```

```
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break
```

This section contains the main loop that processes frames from the video stream, performs face detection and mask detection, and displays the results.

The loop reads frames from the video stream and resizes each frame to have a maximum width of 400 pixels using the `imutils.resize` function.

For each frame, the `detect_and_predict_mask` function is called to detect faces and predict mask/no mask labels. The face locations (`locs`) and corresponding predictions (`preds`) are obtained.

A loop iterates over the face locations and their corresponding predictions. For each face, the bounding box coordinates and predictions are unpacked. The label ("Mask" or "No Mask") and the corresponding color (green for "Mask", red for "No Mask") are determined based on the prediction probabilities.

The label text is formatted with the prediction probability, and it is displayed on the frame using the `cv2.putText` function. Additionally, a rectangle is drawn around the face region using the `cv2.rectangle` function.

The output frame with the labels and bounding boxes is displayed using `cv2.imshow`.

The loop continues until the user presses the "q" key, at which point the loop breaks.

Cleaning up

```
cv2.destroyAllWindows()
vs.stop()
```

After the main loop, the OpenCV windows are closed, and the video stream is stopped and released.