Lebanese University

Mini Project

Depp Learning

Face Mask Detection

Author
Hassan Ismail 5439
Hassan Rammal 5511

Supervisor
Dr. Mohamad Aoude

July 23, 2021

List of Figures

Contents

1 introduction

In this project,We'll show how to use OpenCV, Keras/TensorFlow, and Deep Learning to train a COVID-19 face mask detector.
If deployed correctly, the COVID-19 mask detector we're building here today could potentially be used to help ensure your safety and the safety of others.
additionaly,we'll discuss our COVID-19 face mask detector, detailing how our computer vision/deep learning pipeline will be implemented. From there, we'll review the data set that will be used to train our custom face mask detector.

2 principle of working

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.

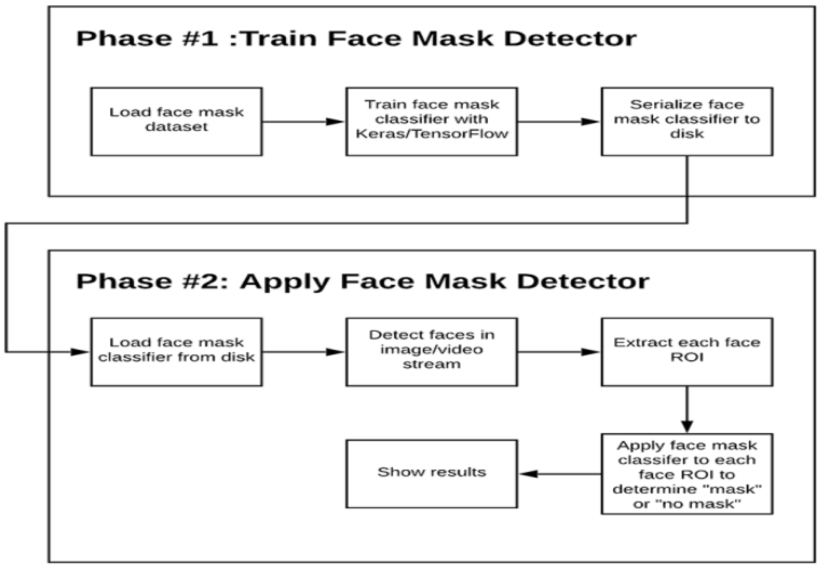

Phase #1 :Train Face Mask Detector
Load face mask dataset
Train face mask classifier with Keras/TensorFlow
Serialize face mask classifier to disk
Phase #2: Apply Face Mask Detector
Load face mask classifier from disk
Detect faces in image/video stream
Extract each face ROI
Apply face mask classifer to each face ROI to determine "mask" or "no mask"
Show results

Figure 1: algorithm of the operation

2.1 training

We will focus on loading our face mask detection dataset from disk, training a model on it (using Keras/TensorFlow), and serializing the face mask detector to disk in this section.

2.2 Deployment

We may then load the mask detector, perform face detection, and categorize each face as having a mask or not having a mask once the face mask detector has been trained.

2.3 Dataset

To begin with our machine learning algorithm, we'll need a dataset to assist us assess and train our computer how to tell the difference between a face wearing a mask and one that isn't. Our dataset consists of 1,376 images belonging to

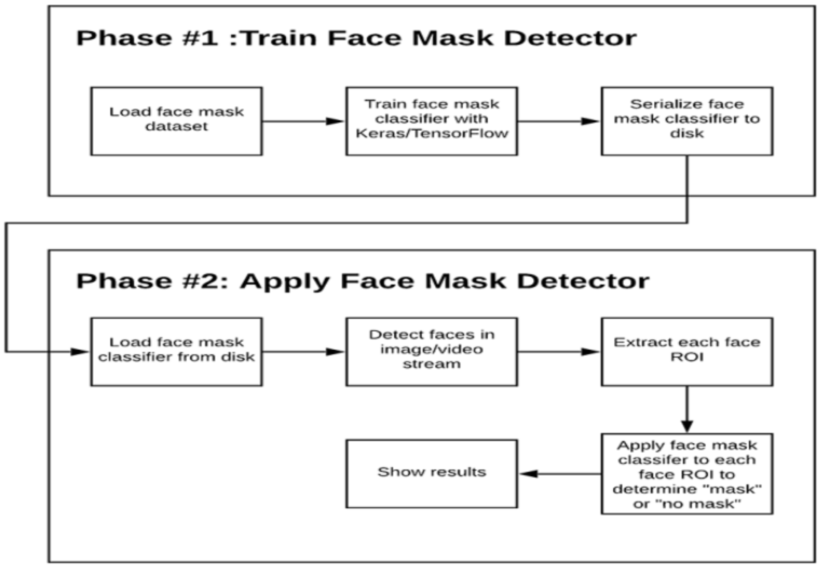

Phase #1 :Train Face Mask Detector
Load face mask dataset
Train face mask classifier with Keras/TensorFlow
Serialize face mask classifier to disk
Phase #2: Apply Face Mask Detector
Load face mask classifier from disk
Detect faces in image/video stream
Extract each face ROI
Apply face mask classifer to each face ROI to determine "mask" or "no mask"
Show results

Figure 2: Dataset with and without mask

two classes:

1. With mask: 690 images
2. Without mask: 686 images

Our objective is to develop an unique deep learning model that can detect whether someone is wearing a mask or not. We can rely on directly obtaining Google pictures, mining the photographs using a specific algorithm, or, in our instance, collecting a dataset from the internet to gather a dataset for a project. Facial landmarks allow us to automatically infer the location of facial structures, including:Eyes,Eyebrows, Nose, Mouth, Jaw line.
To construct a dataset of faces wearing face masks using facial landmarks, we

must first start with a picture of a person who is not wearing a face mask:

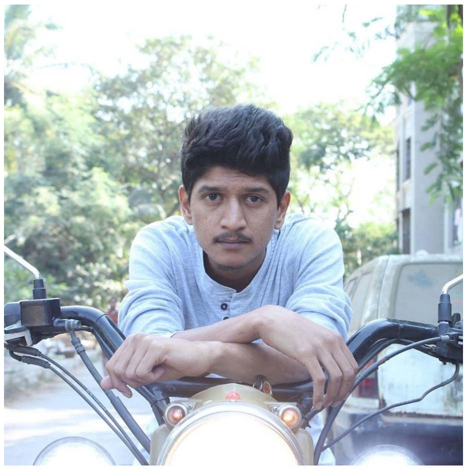


Figure 3: face without mask

Then we use face detection to compute the bounding box position of the face in the picture, and we can extract the face Region of Interest once we know where the face is in the image (ROI).

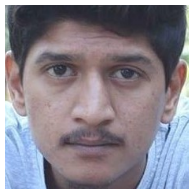


Figure 4: face region

Then we use face markers to locate the eyes, nose, mouth, and other features.

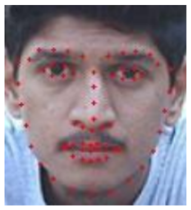


Figure 5: face features

After that, we'll apply a mask to the face we found in order to include it in the dataset.

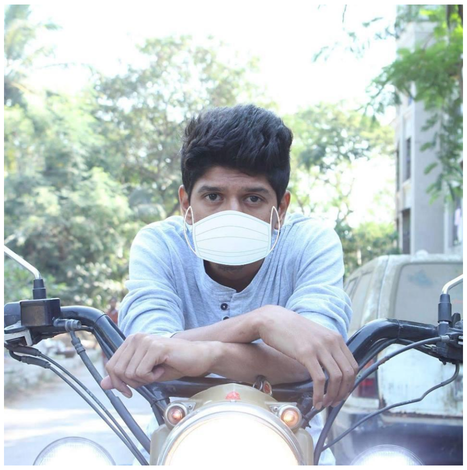


Figure 6: with face mask

We apply this technique on our entire images, this way we create an artificial face mask dataset.

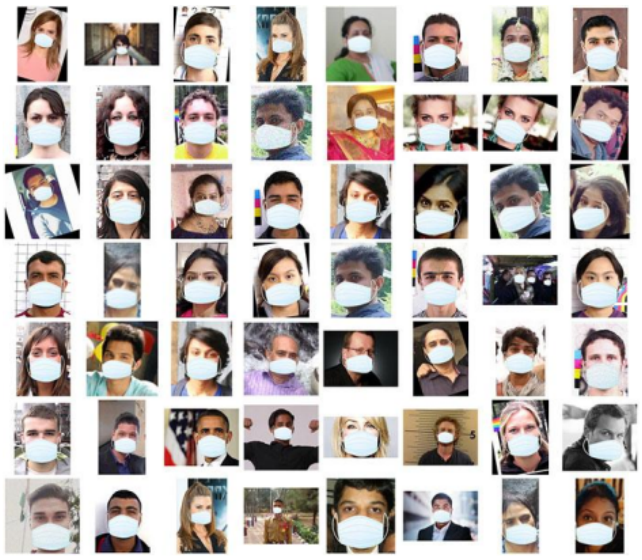
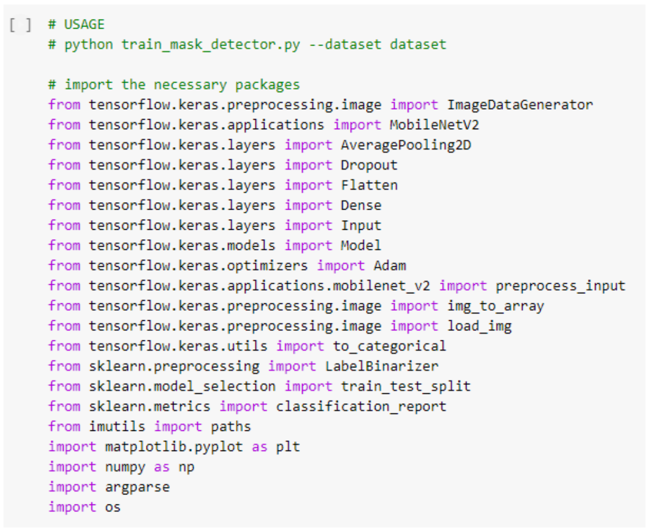


Figure 7: face mask dataset

3 coding

Now, we need to start by importing the needed libraries:

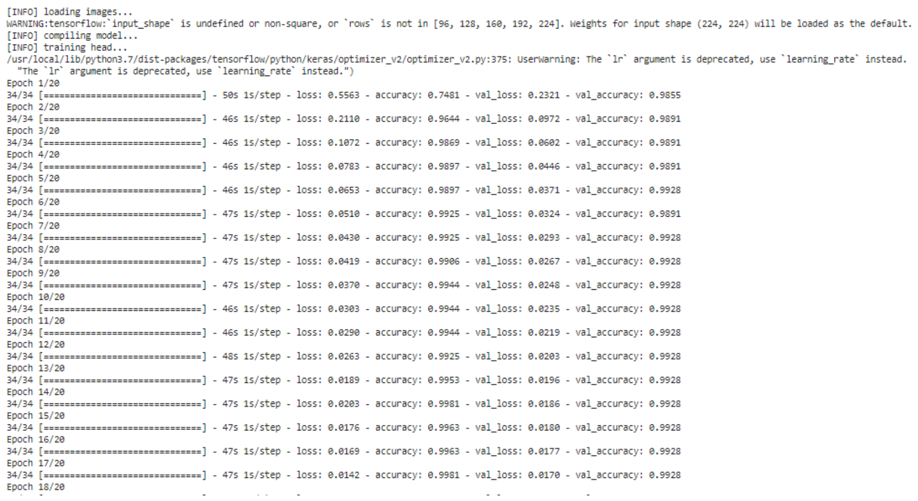


```
[ ] # USAGE
    # python train_mask_detector.py --dataset dataset

    # 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
```

Figure 8: code example

Then after we implement the entire code, we run our program. The program trains the dataset and gives “weight” for each image.



```
[INFO] loading images...
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
[INFO] compiling model...
[INFO] training head...
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
Epoch 1/20
34/34 [==============================] - 50s 1s/step - loss: 0.5563 - accuracy: 0.7481 - val_loss: 0.2321 - val_accuracy: 0.9855
Epoch 2/20
34/34 [==============================] - 46s 1s/step - loss: 0.2110 - accuracy: 0.9644 - val_loss: 0.0972 - val_accuracy: 0.9891
Epoch 3/20
34/34 [==============================] - 46s 1s/step - loss: 0.1072 - accuracy: 0.9869 - val_loss: 0.0602 - val_accuracy: 0.9891
Epoch 4/20
34/34 [==============================] - 46s 1s/step - loss: 0.0783 - accuracy: 0.9897 - val_loss: 0.0446 - val_accuracy: 0.9891
Epoch 5/20
34/34 [==============================] - 46s 1s/step - loss: 0.0653 - accuracy: 0.9897 - val_loss: 0.0371 - val_accuracy: 0.9928
Epoch 6/20
34/34 [==============================] - 47s 1s/step - loss: 0.0510 - accuracy: 0.9925 - val_loss: 0.0324 - val_accuracy: 0.9891
Epoch 7/20
34/34 [==============================] - 47s 1s/step - loss: 0.0430 - accuracy: 0.9925 - val_loss: 0.0293 - val_accuracy: 0.9928
Epoch 8/20
34/34 [==============================] - 47s 1s/step - loss: 0.0419 - accuracy: 0.9906 - val_loss: 0.0267 - val_accuracy: 0.9928
Epoch 9/20
34/34 [==============================] - 47s 1s/step - loss: 0.0370 - accuracy: 0.9944 - val_loss: 0.0248 - val_accuracy: 0.9928
Epoch 10/20
34/34 [==============================] - 46s 1s/step - loss: 0.0303 - accuracy: 0.9944 - val_loss: 0.0235 - val_accuracy: 0.9928
Epoch 11/20
34/34 [==============================] - 46s 1s/step - loss: 0.0290 - accuracy: 0.9944 - val_loss: 0.0219 - val_accuracy: 0.9928
Epoch 12/20
34/34 [==============================] - 48s 1s/step - loss: 0.0263 - accuracy: 0.9925 - val_loss: 0.0203 - val_accuracy: 0.9928
Epoch 13/20
34/34 [==============================] - 47s 1s/step - loss: 0.0189 - accuracy: 0.9953 - val_loss: 0.0196 - val_accuracy: 0.9928
Epoch 14/20
34/34 [==============================] - 47s 1s/step - loss: 0.0203 - accuracy: 0.9981 - val_loss: 0.0186 - val_accuracy: 0.9928
Epoch 15/20
34/34 [==============================] - 47s 1s/step - loss: 0.0176 - accuracy: 0.9963 - val_loss: 0.0180 - val_accuracy: 0.9928
Epoch 16/20
34/34 [==============================] - 47s 1s/step - loss: 0.0169 - accuracy: 0.9963 - val_loss: 0.0177 - val_accuracy: 0.9928
Epoch 17/20
34/34 [==============================] - 47s 1s/step - loss: 0.0142 - accuracy: 0.9981 - val_loss: 0.0170 - val_accuracy: 0.9928
Epoch 18/20
```

Figure 9: code running example

After running the code, we are obtained >99% accuracy on our test set. Looking at Figure 10, we can see there are little signs of over fitting, 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.

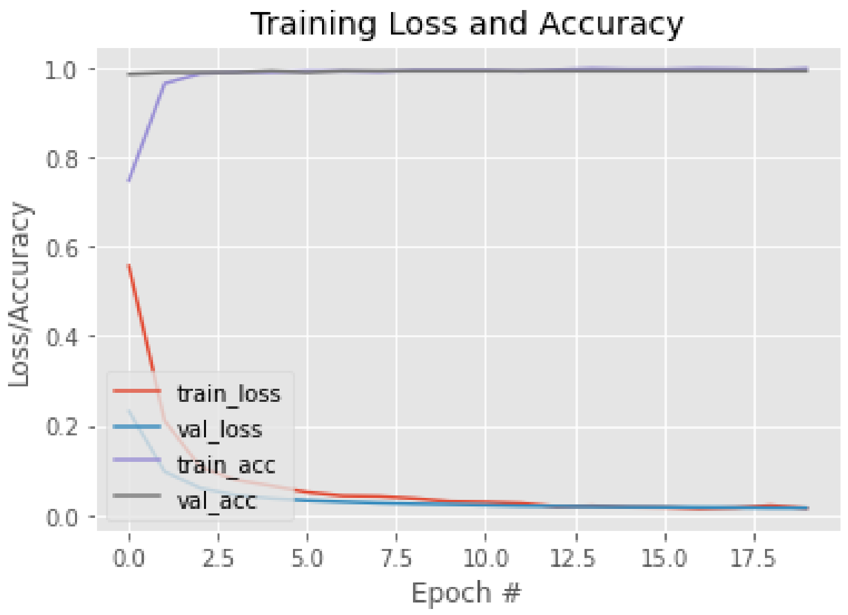


Figure 10: training accuracy

Now that we've established that our set has been properly trained, we can apply our code to the examples supplied or submit an image to evaluate the results.
We decided to test our code on two images of one of us, one with a face mask on and the other without.
The code worked wonderfully, and the system could detect whether the person was wearing a face mask or not.