Face Mask Detection In Sierra Leone Using Machine Learning Techniques

by

Sahr B. Sesay

University Of Management And Technology

Dive Into Code Machine Learning Course

2021

Self-introduction

- **Name**
- **Educational Background**
- **Other Activities**

Introduction / Background

This project is about making a classifier that can differentiate between faces, with masks and without masks.

So for creating this classifier, I need a dataset in the form of Images.

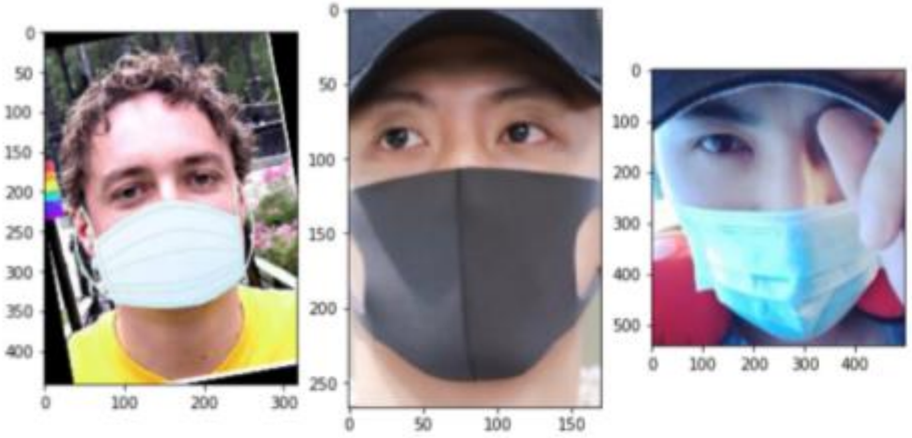
Luckily we have a dataset containing image faces with mask and without a mask.

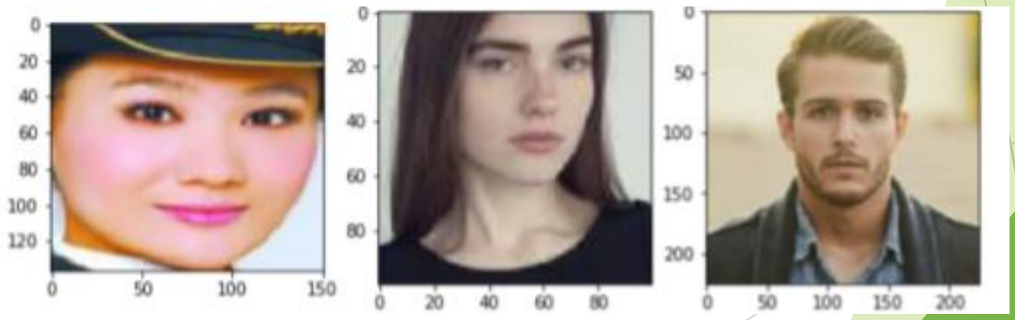
Since these images are very less in number, I cannot train a neural network from scratch. Instead, I will finetune a pre-trained network called MobileNetV2 which is trained on the ImageNet dataset.

Data understanding

This dataset, which is available on Kaggle, has two classes which include people who wear masks and people who do not wear masks. The dataset consists of 1912 images of faces with masks, and 1918 images of faces without masks, which gives us a total of 3830 face images. All images in this dataset have three color channels (RGB).

Below are samples images of people who wear face masks and images of people who do not wear face masks.





MODELING

Let us first import all the necessary libraries we are going to need.

```
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 imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

The next step is to read all the images and assign them to some list. Here I get all the paths associated with these images and then label them accordingly. Remember the dataset is contained in two folders labeled- with masks and without masks. So we can easily get the labels by extracting the folder name from the path. Also, we preprocess the image and resize it to 224x 224 dimensions.

```
imagePaths = list(paths.list_images('/content/drive/MyDrive/Colab Notebooks/Graduation_Assignment/
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)
```

The next step is to load the pre-trained model and customize it according to the problem. So I remove the top layers of this pre-trained model and add few layers. As you can see the last layer has two nodes as we have only two outputs. This is called transfer learning.

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_shape=(224, 224, 3))
# construct the head of the model that will be placed on top of 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 trai
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
for layer in baseModel.layers:
    layer.trainable = False
```

Now I need to convert the labels into one-hot encoding. After that, I split the data into training and testing sets to evaluate them. Also, the next step is data augmentation which significantly increases the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, rotation, shearing and horizontal flipping are commonly used to train large neural networks.

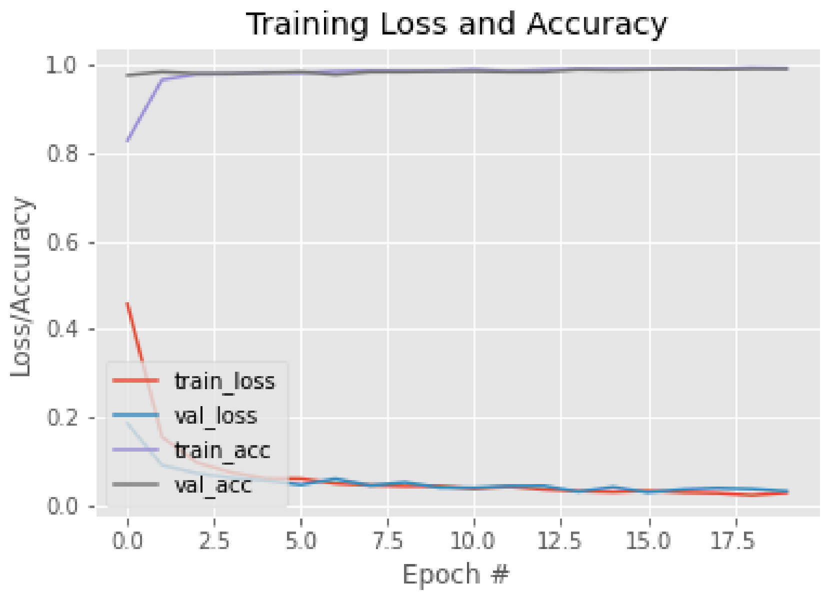
```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, 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")
```

The next step is to compile the model and train it on the augmented data.

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
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)
```

Now that the model is trained, I will plot a graph to see its learning curve.

```
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("plot.png")
```



Also, we save the model for later use.

```
#To save the trained model
model.save('mask_recognition_model.h5')
```

How to do Real-time Mask detection

Now that the model is trained, I can modify the code in the first section so that it can detect faces and also tell if the person is wearing a mask or not.

In order for this mask detector model to work, it needs images of faces.

For this, I will detect the frames with faces using the methods as shown in the first section and then pass them to our model after preprocessing them. So let us first import all the libraries we need.

```
import cv2
import os
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
import numpy as np

```

Real time Mask detection

I will be using Haar Cascade algorithm, also known as Voila-Jones algorithm to detect faces. It is basically a machine learning object detection algorithm which is used to identify objects in an image or video.

In OpenCV, there are several trained Haar Cascade models which are saved as XML files. Instead of creating and training the model from scratch, I will use this file. I am going to use “haarcascade_frontalface_alt2.xml” file in this project.

Now let’s start coding this up……………………………………………..

The first step is to find the path to the “haarcascade_frontalface_alt2.xml” file. I do this by using the OS module of Python language.

The next step is to load the classifier. The path to the above XML file goes as an argument to Cascade Classifier() method of OpenCV.

```
# Model for face detection
cascPath = os.path.dirname(
    cv2.__file__) + "/data/haarcascade_frontalface_defult.xml"
faceCascade = cv2.CascadeClassifier(cascPath)
# import the mask detection model
model = load_model("mask_recognition_model.h5")

video_capture = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(30, 30),
                                         flags=cv2.CASCADE_SCALE_IMAGE
                                        )
```

Next, I define some lists.
The faces list contains all the faces that are detected by the face Cascade model and the preds list is used to store the predictions made by the mask detector model.

```
faces_list=[]
preds=[]
```

Also since the faces variable contains the top-left corner coordinates, height and width of the rectangle encompassing the faces, i can use that to get a frame of the face and then preprocess that frame so that it can be fed into the model for prediction.

The preprocessing steps are same that are followed when training the model in the second section. For example, the model is trained on RGB images so we convert the image into RGB here

```
    for (x, y, w, h) in faces:
        face_frame = frame[y:y+h,x:x+w]
        face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
        face_frame = cv2.resize(face_frame, (224, 224))
        face_frame = img_to_array(face_frame)
        face_frame = np.expand_dims(face_frame, axis=0)
        face_frame =  preprocess_input(face_frame)
        faces_list.append(face_frame)
        if len(faces_list)>0:
            preds = model.predict(faces_list)
```

After getting the predictions, we draw a rectangle over the face and put a label according to the predictions.

```
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
        cv2.putText(frame, label, (x, y- 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
```

Next, I just display the resulting frame and also set a way to exit this infinite loop and close the video feed.

By pressing the ‘q’ key, we can exit the script here

```
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

Evaluation

This brings us to the end of this project where we learned how to detect faces in real-time and also designed a model that can detect faces with masks. Using this model I was able able to modify the face detector to mask detector.

The model was trained in Google Collab and later imported int the Real time face mask detection code on Jupiter notebook.

The dataset using in the project is not from Sierra Leone because such dataset is not available at the moment and it take a lot of time and resources to create one , but the dataset was rather making from Kaggle.

The dataset fit perfectly into the project become it has to deal with classifying between people wearing a face mask and those that do not.