# 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 os

# initialize the initial learning rate, number of epochs to train for,

# and batch size

INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

DIRECTORY = r'/content/drive/MyDrive/cv/dataset'

CATEGORIES = ["with\_mask", "without\_mask"]

# grab the list of images in our dataset directory, then initialize

# the list of data (i.e., images) and class images

print("[INFO] loading images...")

data = []

labels = []

for category in CATEGORIES:

    path = os.path.join(DIRECTORY, category)

    for img in os.listdir(path):

      img\_path = os.path.join(path, img)

      image = load\_img(img\_path, target\_size=(224, 224))

      image = img\_to\_array(image)

      image = preprocess\_input(image)

      data.append(image)

      labels.append(category)

# perform one-hot encoding on the labels

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

labels = to\_categorical(labels)

data = np.array(data, dtype="float32")

labels = np.array(labels)

(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")

# load the MobileNetV2 network, ensuring the head FC layer sets are

# left off

baseModel = MobileNetV2(weights="imagenet", include\_top=False,

  input\_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the

# 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 train)

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 first training process

for layer in baseModel.layers:

  layer.trainable = False

# compile our model

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)

# make predictions on the testing set

print("[INFO] evaluating network...")

predIdxs = model.predict(testX, batch\_size=BS)

# for each image in the testing set we need to find the index of the

# label with corresponding largest predicted probability

predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report

print(classification\_report(testY.argmax(axis=1), predIdxs,

  target\_names=lb.classes\_))

# serialize the model to disk

print("[INFO] saving mask detector model...")

model.save("mask\_detector.model", save\_format="h5")

# plot the training loss and accuracy

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")