```
# USAGE
# python train.py --dataset dataset

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, InceptionV3
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout, Conv2D, BatchNormalization, MaxPooling2D
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.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, precision_score
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
```

```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-3
EPOCHS = 25
BS = 10
```

```
# 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...")
imagePaths = list(paths.list_images("/content/drive/My Drive/keras-covid-19/dataset"))
data = []
labels = []
```

```
[INFO] loading images...
```

```
# 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 image, swap color channels, and resize it to be a fixed
  # 224x224 pixels while ignoring aspect ratio
  image = cv2.imread(imagePath)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image = cv2.resize(image, (224, 224))
```

```
  # update the data and labels lists, respectively
  data.append(image)
  labels.append(label)
```

```
# convert the data and labels to NumPy arrays while scaling the pixel
# intensities to the range [0, 255]
data = np.array(data) / 255.0
labels = np.array(labels)
```

```
# perform one-hot encoding on the labels
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=0.50, stratify=labels, random_state=42)
print(len(trainX), len(trainY), len(testX), len(testY))
```

```
125 125 125 125
```

```
# initialize the training data augmentation object
trainAug = ImageDataGenerator(
  rotation_range=15,
  fill_mode="nearest")
```

```
# load the VGG16 network, ensuring the head FC layer sets are left
# off
baseModel = InceptionV3(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=(4, 4))(headModel)
#headModel = Conv2D(32, activation="relu")(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = BatchNormalization()(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(64, activation="softplus")(headModel)
headModel = BatchNormalization()(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(128, activation='relu')(headModel)
headModel = BatchNormalization()(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
#headModel.summary()
```

```
# 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"])
```

```
[INFO] compiling model...
```

```
# train the head of the network
print("[INFO] training head...")
H = model.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

```
[INFO] training head...
Epoch 1/25
12/12 [==============================] - 3s 283ms/step - loss: 1.0924 - accuracy: 0.5826 - val_los
Epoch 2/25
12/12 [==============================] - 2s 127ms/step - loss: 0.8306 - accuracy: 0.6435 - val_los
Epoch 3/25
12/12 [==============================] - 2s 128ms/step - loss: 0.7994 - accuracy: 0.6261 - val_los
Epoch 4/25
12/12 [==============================] - 2s 128ms/step - loss: 0.4940 - accuracy: 0.7739 - val_los
Epoch 5/25
12/12 [==============================] - 2s 132ms/step - loss: 0.6465 - accuracy: 0.6696 - val_los
Epoch 6/25
12/12 [==============================] - 2s 126ms/step - loss: 0.5355 - accuracy: 0.7565 - val_los
Epoch 7/25
12/12 [==============================] - 2s 128ms/step - loss: 0.5480 - accuracy: 0.8087 - val_los
Epoch 8/25
12/12 [==============================] - 2s 126ms/step - loss: 0.4401 - accuracy: 0.8174 - val_los
Epoch 9/25
12/12 [==============================] - 2s 126ms/step - loss: 0.2884 - accuracy: 0.8957 - val_los
Epoch 10/25
12/12 [==============================] - 2s 128ms/step - loss: 0.4164 - accuracy: 0.8348 - val_los
Epoch 11/25
12/12 [==============================] - 2s 126ms/step - loss: 0.3365 - accuracy: 0.8609 - val_los
Epoch 12/25
12/12 [==============================] - 2s 126ms/step - loss: 0.2993 - accuracy: 0.8696 - val_los
```

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
```

```
[INFO] evaluating network...
```

```
# 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)
```

```
Epoch 19/25
```

```
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
  target_names=lb.classes_))
```

```
              precision    recall  f1-score   support

       covid       0.92      0.97      0.94        62
      normal       0.97      0.92      0.94        63

    accuracy                           0.94       125
   macro avg       0.94      0.94      0.94       125
weighted avg       0.95      0.94      0.94       125
```

```
# compute the confusion matrix and and use it to derive the raw
# accuracy, sensitivity, and specificity
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
```

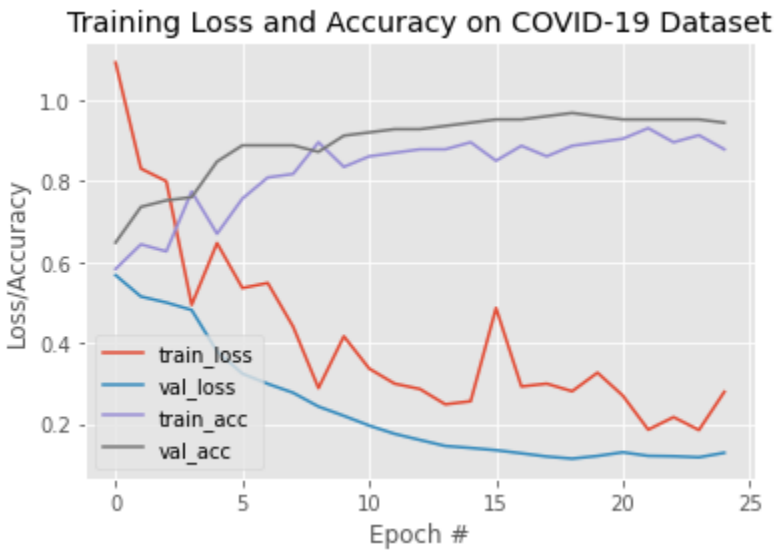
```
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
#pre = precision_score(testY,trainY)
```

```
# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[60  2]
 [ 5 58]]
acc: 0.9440
sensitivity: 0.9677
specificity: 0.9206
```

```
# 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 on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
#plt.savefig(args["plot"])
```

```
<matplotlib.legend.Legend at 0x7fe8039502e8>
```




```
# serialize the model to disk
print("[INFO] saving COVID-19 detector model...")
#model.save(args["model"], save_format="h5")
```