FACE MASK DETECTION USING TENSORFLOW

TABLE OF CONTENT

- INTRODUCTION
- AIMS AND OBJECTIVES
- SYSTEM REQUIREMENTS STUDY
    - SOFTWARE REQUIREMENTS
    - REQUIRED PACKAGES
- METHODOLOGY
- CONCLUSION
- REFERENCES

INTRODUCTION

- The world is fighting with Covid19 pandemic. There are so many essential equipment needed to fight against Corona virus. One of such the most essential one is Face Mask.

- Wearing a face mask will help prevent the spread of infection and prevent the individual from contracting any airborne infectious germs. When someone coughs, talks, sneezes they could release germs into the air that may infect others nearby. Face masks are part of an infection control strategy to eliminate cross-contamination.

AIMS AND OBJECTIVES

- Our aim is to train machine learning algorithm of face mask detection on our dataset using Keras and TensorFlow.

- Given the trained COVID-19 face mask detector, we’ll proceed to implement two more additional Python scripts used to:

1. Detect COVID-19 face masks in *images*
2. Detect face masks in *real-time video streams*

METHODOLOGY

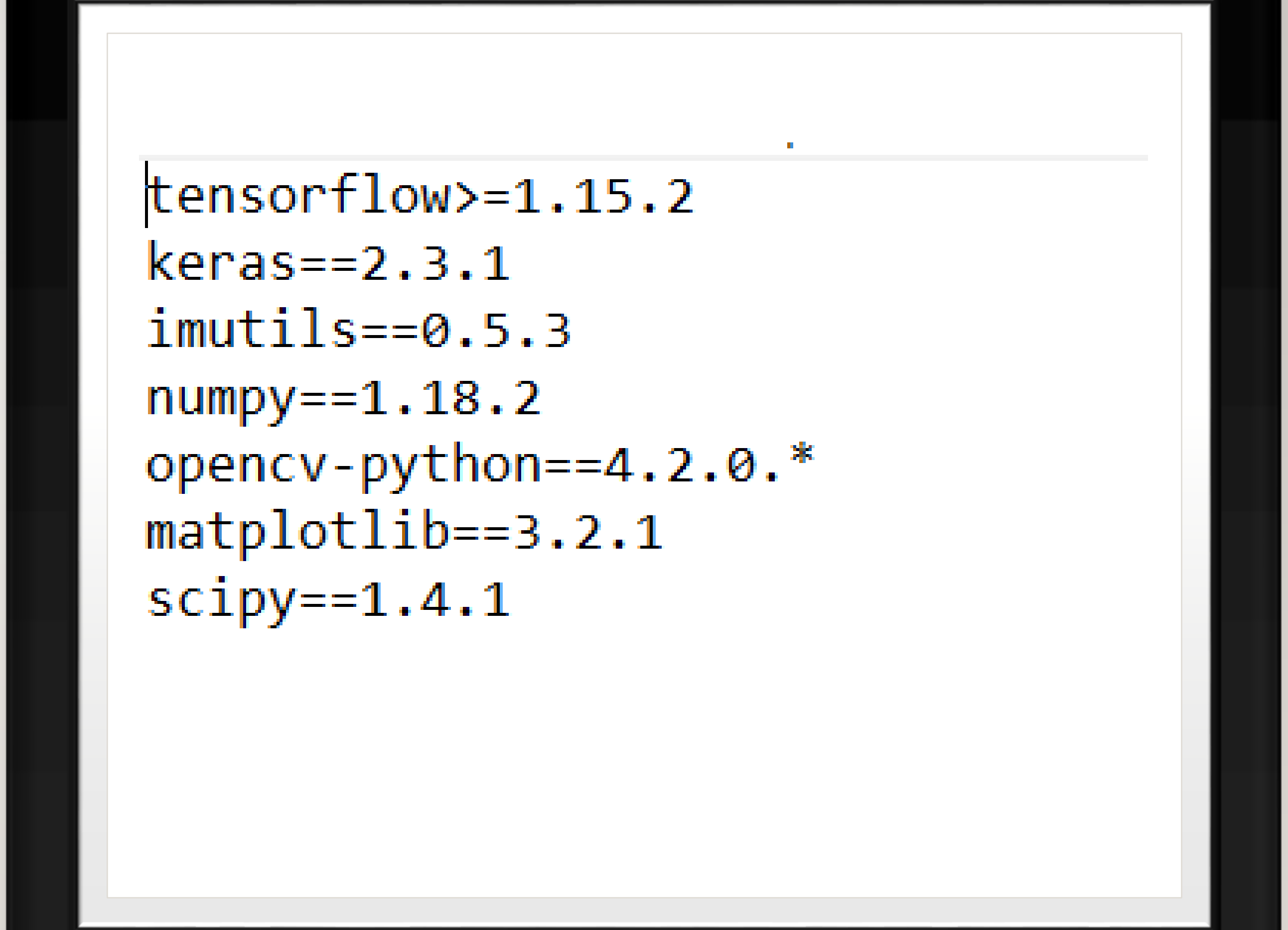
SYSTEM REQUIREMENTS STUDY

- Software requirements

  Processor – Intel Core i5 CPU

  RAM- 8.0 GB

- Required Packages



```
tensorflow>=1.15.2
keras==2.3.1
imutils==0.5.3
numpy==1.18.2
opencv-python==4.2.0.*
matplotlib==3.2.1
scipy==1.4.1
```

Ⅰ. DATA VISUALIZATION

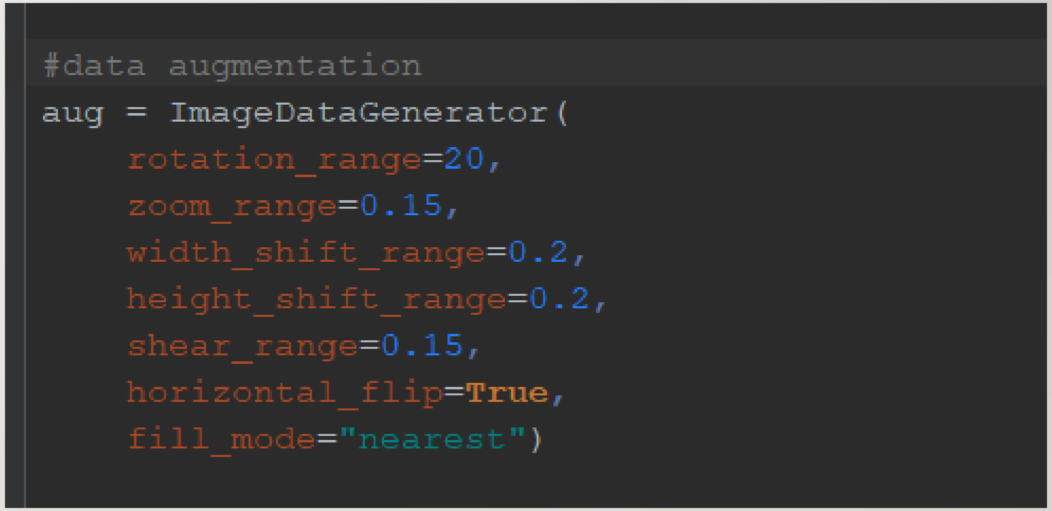
- In the first step, let us visualize the total number of images in our dataset in both categories. We can see that there are ***690*** images in the ‘***yes***’ class and ***686*** images in the ‘***no***’ class.

- ‘yes’ indicates images with masks

- ‘no’ indicates images without masks

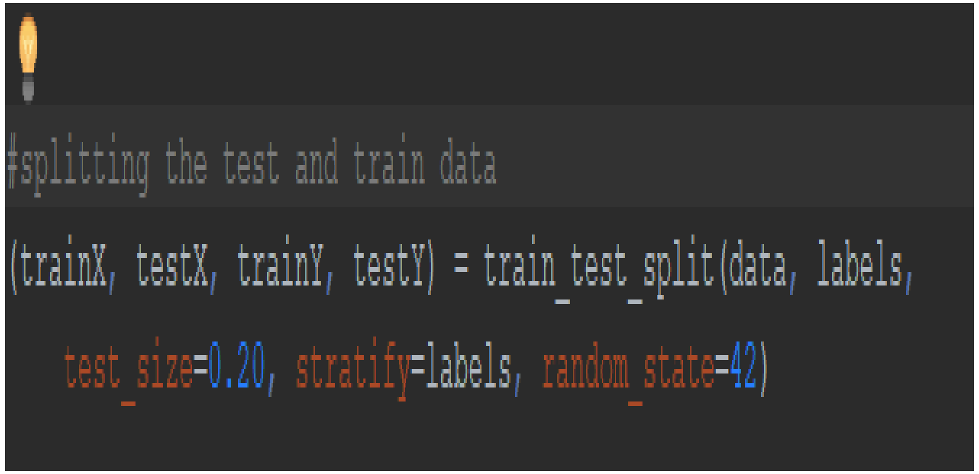
2 DATA AUGMENTATION

- In the next step, we ***augment*** our dataset to include more number of images for our training. In this step of *data augmentation*, we *rotate* and *flip* each of the images in our dataset. We see that, after data augmentation, we have a total of ***2751*** images with ***1380*** images in the ‘***yes***’ class and ‘***1371***’ images in the ‘***no***’ class.



```
#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")
```

3. SPLITTING THE DATA

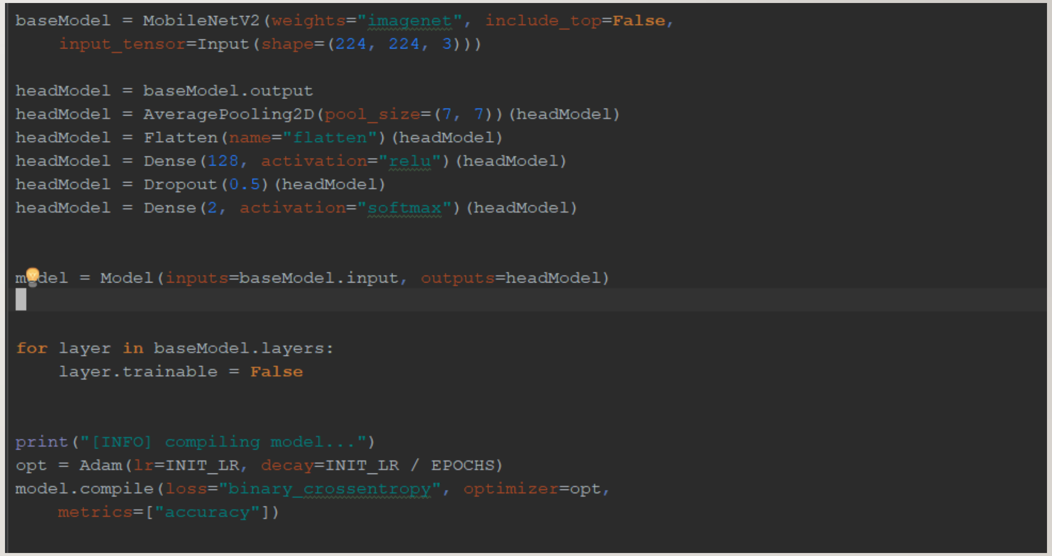


```
#splitting the test and train data
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
```

- In this step, we ***split*** our data into the ***training set*** which will contain the images on which the CNN model will be trained and the ***test set*** with the images on which our model will be tested.
- In this, we take ***split_size =0.8***, which means that ***80%*** of the total images will go to the *training set* and the remaining ***20%*** of the images will go to the *test set.*

4. BUILDING THE MODEL

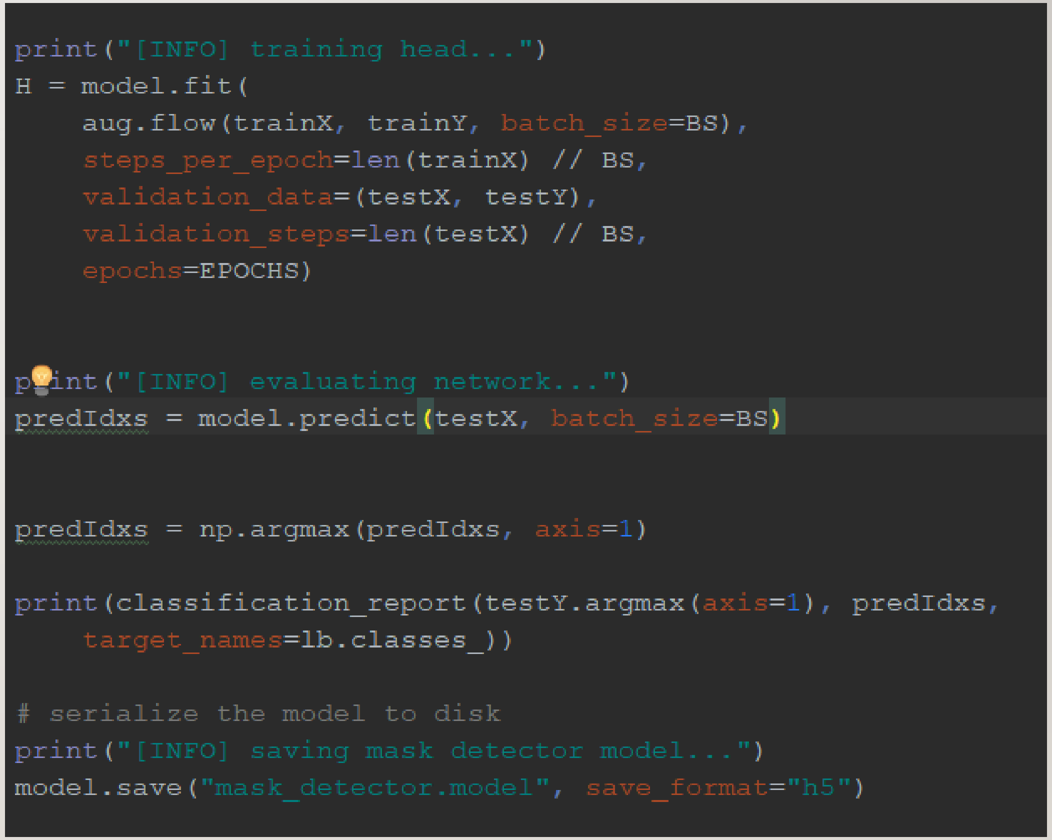
- In the next step, we build our ***depthwise convolution model*** with various layers such as *Conv2D, AveragePooling2D, Flatten, Dropout* and *Dense*. In the last Dense layer, we use the '***softmax***' function to output a vector that gives the *probability* of each of the two classes.



```python
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

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)


model = Model(inputs=baseModel.input, outputs=headModel)


for layer in baseModel.layers:
    layer.trainable = False


print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

5. TRAINING THE MOBILE NET MODEL

- This step is the main step where we fit our images in the training set and the test set to our Sequential model we built using ***keras*** library. I have trained the model for ***20 epochs*** (iterations). However, we can train for more number of epochs to attain higher accuracy lest there occurs *over-fitting*.
- We see that after the 20th epoch, our model has an accuracy of ***87.86%*** with the training set and an accuracy of ***90.19%*** with the test set. This implies that it is well trained without any over-fitting.



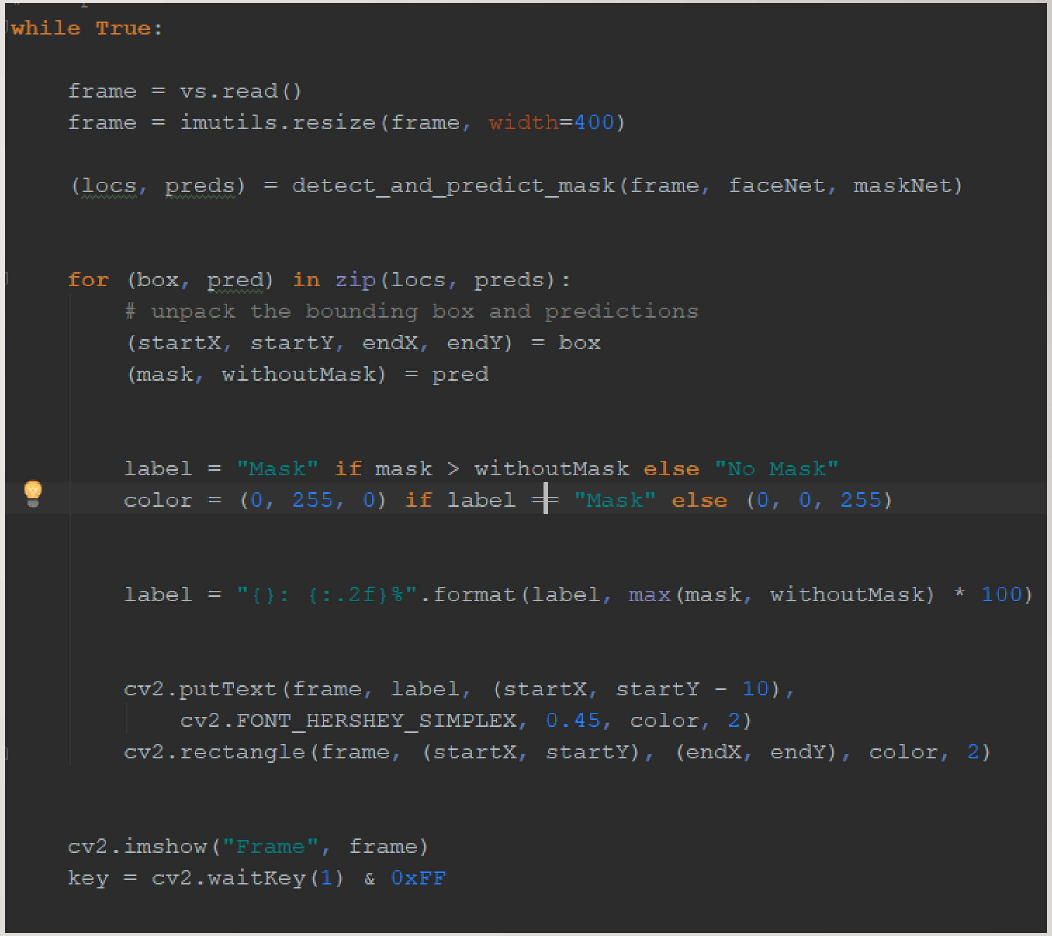
```
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)


print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)


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

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")
```

6. LABELING THE INFORMATION

- After building the model, we label two probabilities for our results. ***[‘0’*** *as* ***‘without_mask’*** *and* ***‘1’*** *as* ***‘with_mask’]***. I am also setting the boundary rectangle color using the RGB values.***[‘RED’*** *for* ***‘without_mask’*** *and* ***‘GREEN’*** *for* ***‘with_mask]***

7. DETECTING THE FACES WITH AND WITHOUT MASKS

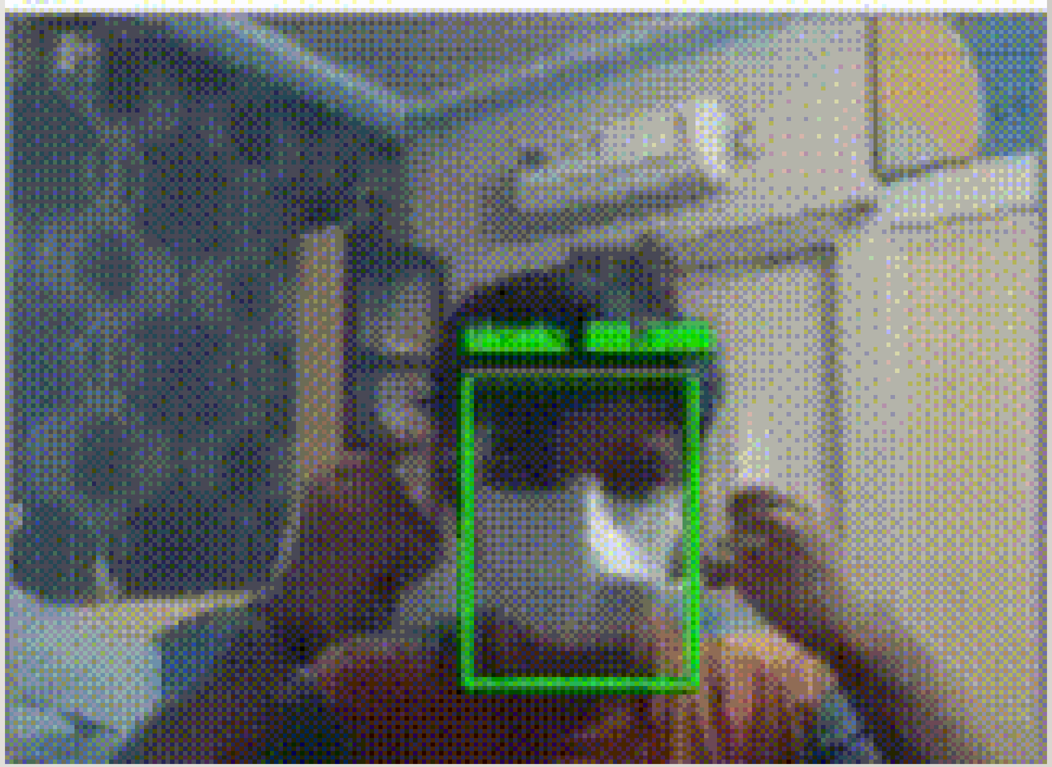
- In the last step, we use the OpenCV library to run an infinite loop to use our web camera in which we detect the face using the *Cascade Classifier*.
- The model will predict the possibility of each of the two classes ([without-mask, with-mask]). Based on which probability is higher, the label will be chosen and displayed around our faces.



```
while True:

    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)


    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred


        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, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)


    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
```

DEMO

- Let us see the demo where I try it out on myself!



CONCLUSION

- The technology assures reliable and real-time face detection of users wearing masks. Besides, the system is easy to deploy into any existing system of a business while keeping the safety and privacy of users’ data. So the face mask detection system is going to be the leading digital solution for most industries, especially retail, healthcare, and corporate sectors.

REFRENCES

- https://www.udemy.com/course/python-for-vision-and-detection-opencv-python

- https://www.tensorflow.org/guide

- https://stackoverflow.com

THANK YOU