Classification of Spoofing Attack Detection using Deep Learning Algorithms - Configuration Manual

MSc Research Project
Data Analytics

Shreya Verma
Student ID: x20229291

School of Computing
National College of
Ireland

Supervisor: Dr. Catherine Mulwa

National College of IrelandProject Submission Sheet School of Computing

Student Name:	Shreya Verma
Student ID:	x20229291
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	15/08/2022
Project Title:	Classification of Spoofing Attack Detection using Deep Learning Algorithms - Configuration Manual
Word Count:	XXX
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keepa copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Classification of Spoofing Attack Detection using Deep Learning Algorithms - Configuration Manual

Shreya Verma
x20229291

1. Introduction

The environmental setup specification to construct the research project ”spoofing attack detection using deep machine learning” is explained in this configuration manual This manual provide the step by step explanation and the pre-requisite for this research project. Specifications for hardware and software are presented in Sections 2 and 3. Section3 deals with data sources. Section 4 deals with execution;

1.1 Project Objective

The main objective of this project to detect the spoofing detection using machine learning in which the dataset contains the two classes which are fake and real image so using deep learning algorithms we have to classify the images into fake or real image using convolution neural network model which are MobileNetV2, Vgg16, ResNet50, EfficientNetB4, and CNN models to classify the images.

2. Configuration setup

The configuration setup needs hardware and software requirement for the research project.To train the model which has the large size of images we need GPU (Graphics Processing Unit) in google colab.

2.1 Hardware Configuration

RAM: 8GB
Processor: Inter core TM i5 CPU
System: Operating system-64 Bit
Operating system: Windows 10 (Microsoft cooperation)
GPU: Google Colab (Tesla K90)

2.2 Software Configuration

The software configuration for this research project is done using the software environment which is Google colab and the Jupiter notebook for the deep learning models. The coding for all the models is done python language and the version used for the python is 3.6.3. There are so many python libraries have been used while carried out this research project. The libraries which are used are Tersonflow (2.5.0), Keras, Matplotlib (3.3.3), sklearn, pandas, numpy and so on.

3. Data Gathering

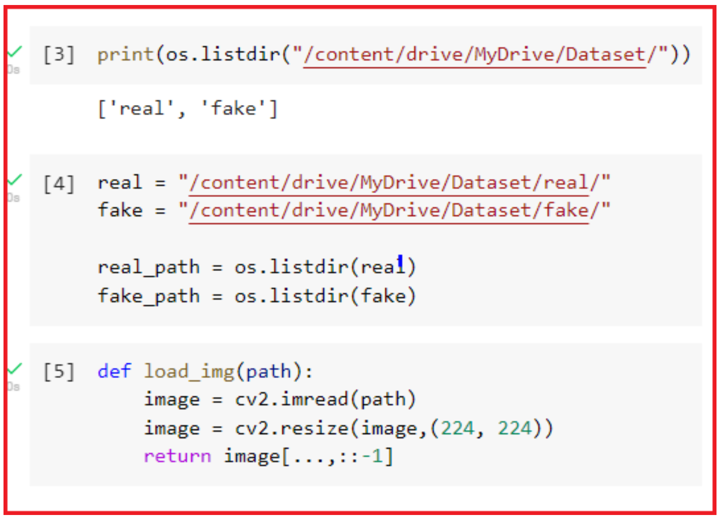
Many websites, like Kaggle, Facebook AI, pictures datasets, and others, offer the open source dataset of images and videos. This research on spoofing attack detection utilizes a publicly accessible dataset of real and fake photos from Kaggle. The collection consists of very high resolution faces of random people in both genuine and fake photographs. It consists of 20,000 high-quality PNG images and pixels in total. The collection contains fictitious images created by Photoshop specialists. The Photoshop tools are used to modify the fraudulent photos. Before starting the data preprocessing and transformation the steps are followed to update the dataset in google colab. First upload the dataset onthe drive and to upload on the google cola run the below command code.

4. Data Transformation

The dataset contains real and fake images and all the images are in .png formate and high quality image, The pixel resolution of each pixel is in the 1024 x 1024. All the images on the drive are stored in the dataset folder and run the code for mounting the data.

5. Data Preprocessing

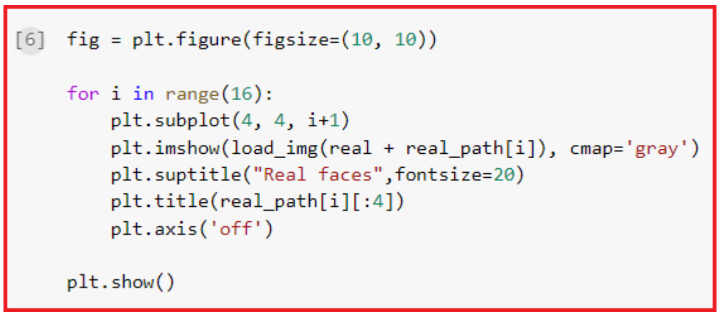
Once the dataset was uploded to the drive and load in the google colab the divide the real and fake images and than load the file path to the realpath and fakepath variable. Once the data is loaded after that resize the images using openCv into 224 X 224 size and return the images into an array.



```
[3]  print(os.listdir("/content/drive/MyDrive/Dataset/"))

     ['real', 'fake']

[4]  real = "/content/drive/MyDrive/Dataset/real/"
     fake = "/content/drive/MyDrive/Dataset/fake/"

     real_path = os.listdir(real)
     fake_path = os.listdir(fake)

[5]  def load_img(path):
         image = cv2.imread(path)
         image = cv2.resize(image,(224, 224))
         return image[...,::-1]
```

Figure 1: Real and Fake Image dataset loaded

The Exploratory data analysis of real and fake images. The below figure presents the random real and fake images from the dataset. The figure size for the images taken is 10 x 10 and for displaying the images the range took is 16 to fetch the 16 images from dataset. The images are plot using the matplotlib library in python.



```
[6]  fig = plt.figure(figsize=(10, 10))

     for i in range(16):
         plt.subplot(4, 4, i+1)
         plt.imshow(load_img(real + real_path[i]), cmap='gray')
         plt.suptitle("Real faces",fontsize=20)
         plt.title(real_path[i][:4])
         plt.axis('off')

     plt.show()
```

Figure 2: Random Real Image fetch from dataset

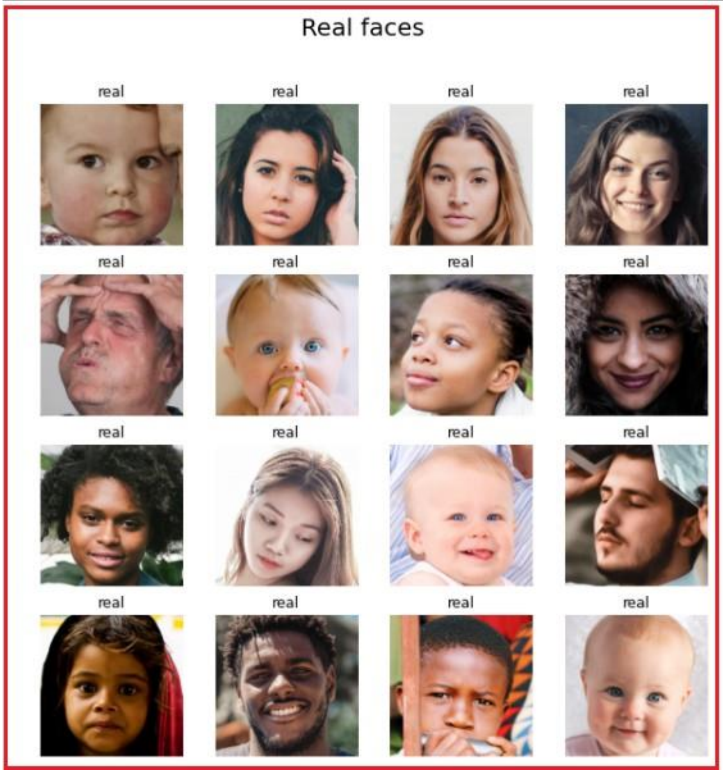
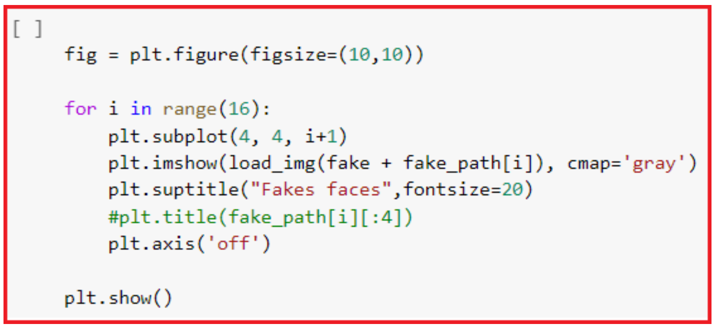


Figure 3: Real Images



```
[ ]
    fig = plt.figure(figsize=(10,10))

    for i in range(16):
        plt.subplot(4, 4, i+1)
        plt.imshow(load_img(fake + fake_path[i]), cmap='gray')
        plt.suptitle("Fakes faces",fontsize=20)
        #plt.title(fake_path[i][:4])
        plt.axis('off')

    plt.show()
```

Figure 4: Random Fake Image fetch from dataset

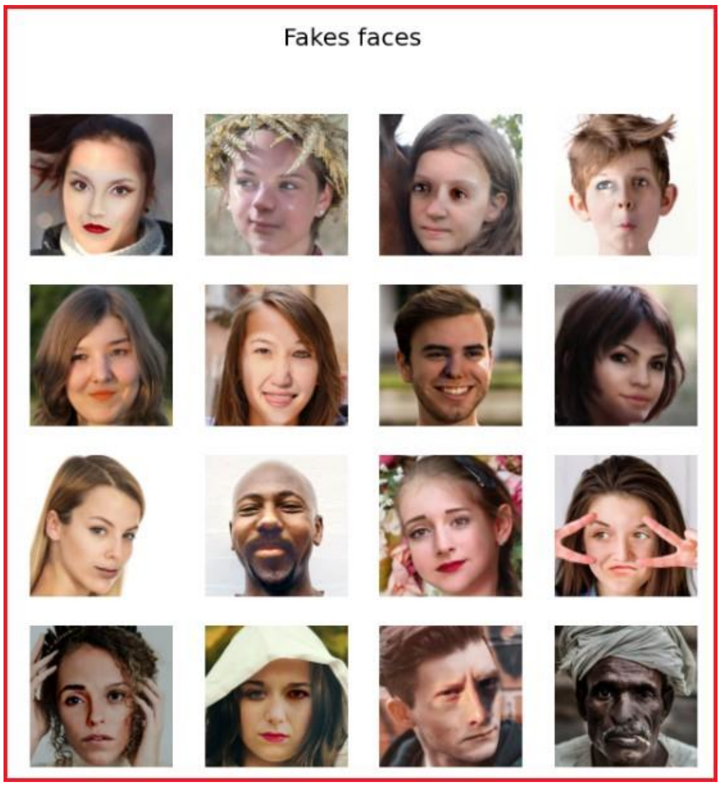
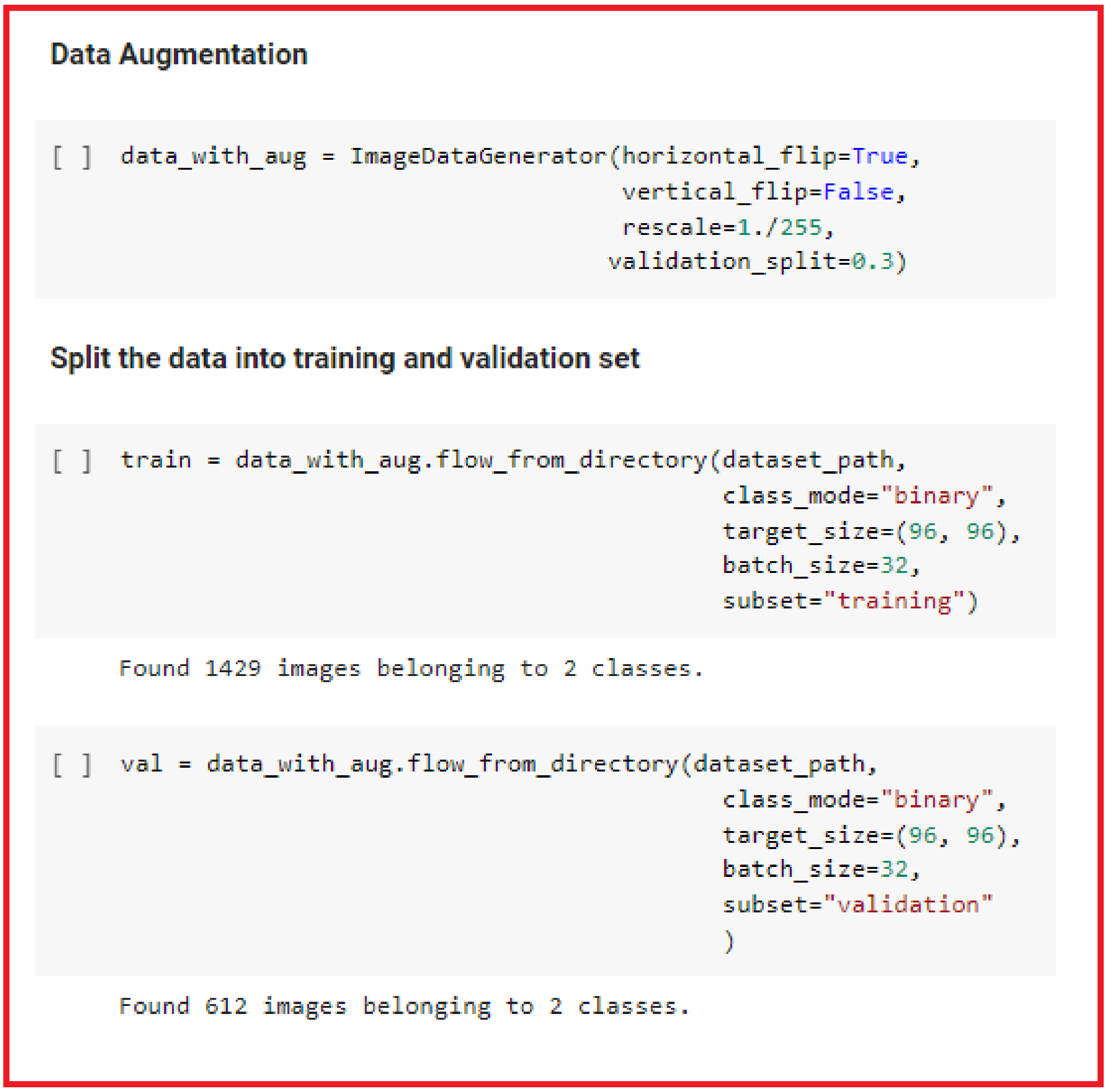


Figure 5: Fake Image

6. Data Augmentation

The data augmentation is used to split the data in training and validation set. The splitting is done on training and validation set is 70% and 30% Using data augmentation function the images are re-scaled, flip horizontal and vertical.



Data Augmentation

```
[ ]  data_with_aug = ImageDataGenerator(horizontal_flip=True,
                                        vertical_flip=False,
                                        rescale=1./255,
                                       validation_split=0.3)
```

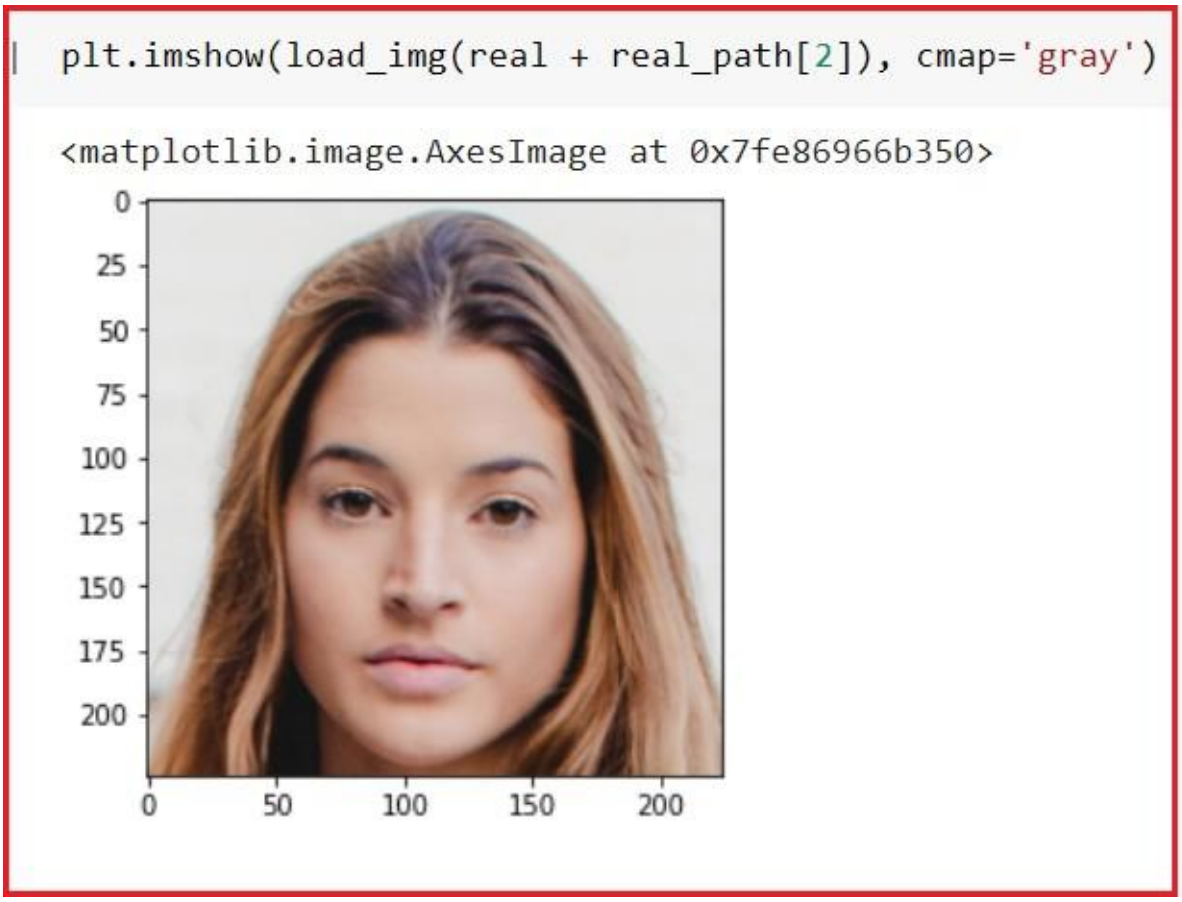
Split the data into training and validation set

```
[ ]  train = data_with_aug.flow_from_directory(dataset_path,
                                               class_mode="binary",
                                               target_size=(96, 96),
                                               batch_size=32,
                                               subset="training")

     Found 1429 images belonging to 2 classes.
```

```
[ ]  val = data_with_aug.flow_from_directory(dataset_path,
                                               class_mode="binary",
                                               target_size=(96, 96),
                                               batch_size=32,
                                               subset="validation"
                                               )

     Found 612 images belonging to 2 classes.
```

Figure 6: Data Augmentation

After data augmentation the plt.imshow function of the python library matplotlib is executed to check the changes in the images. The image is rescale into the 200 X 200.



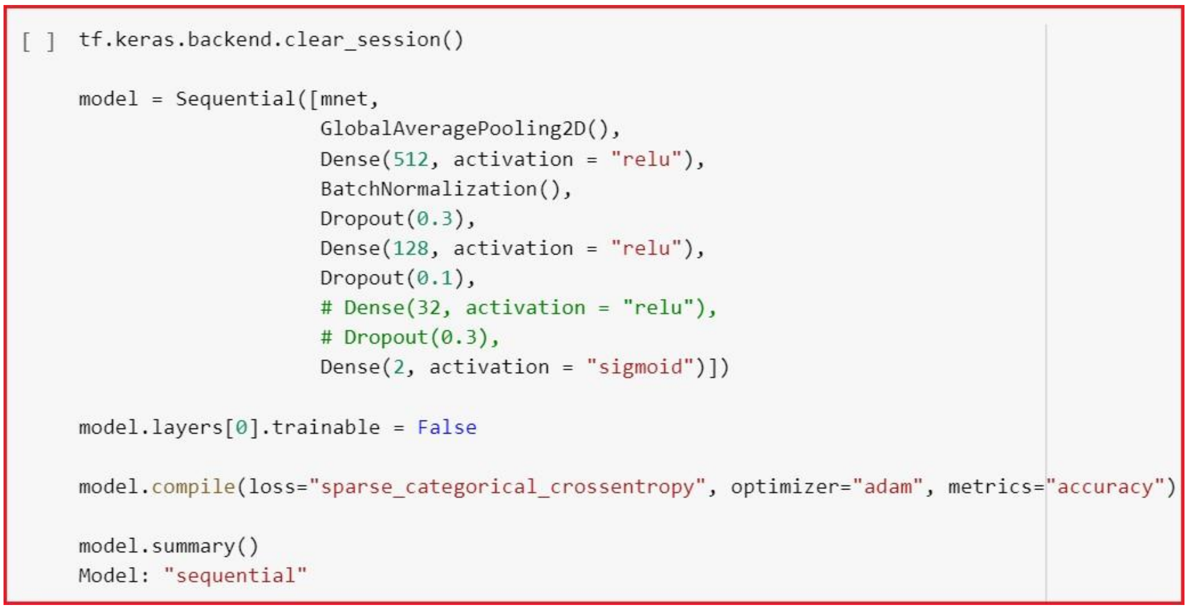
```
plt.imshow(load_img(real + real_path[2]), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fe86966b350>
```

Figure 7: After Data Augmentation the images

7. Implementation of CNN Models

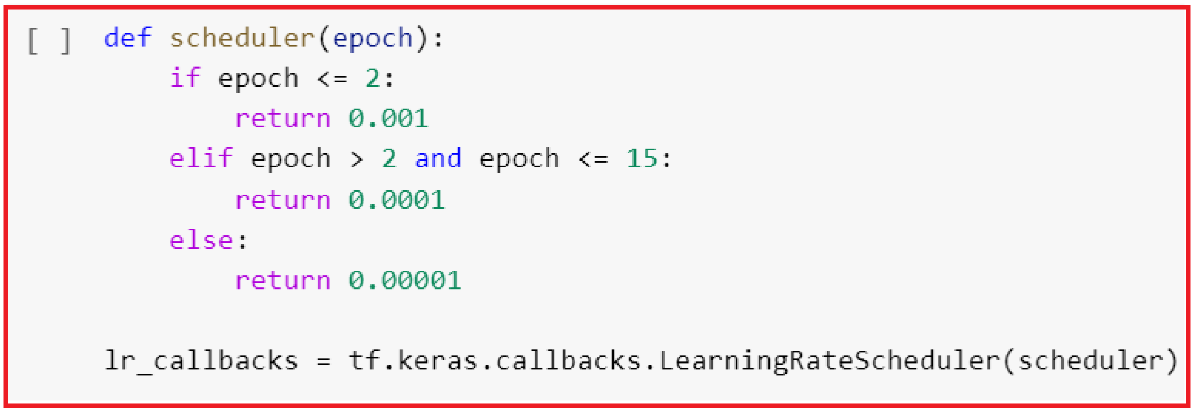
7.1 Implementation and Evaluation of MobileNetV2 (Model 1



```
[ ]  tf.keras.backend.clear_session()

     model = Sequential([mnet,
                         GlobalAveragePooling2D(),
                         Dense(512, activation = "relu"),
                         BatchNormalization(),
                         Dropout(0.3),
                         Dense(128, activation = "relu"),
                         Dropout(0.1),
                         # Dense(32, activation = "relu"),
                         # Dropout(0.3),
                         Dense(2, activation = "sigmoid")])

     model.layers[0].trainable = False

     model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics="accuracy")

     model.summary()
     Model: "sequential"
```

Figure 8: Building Model 1

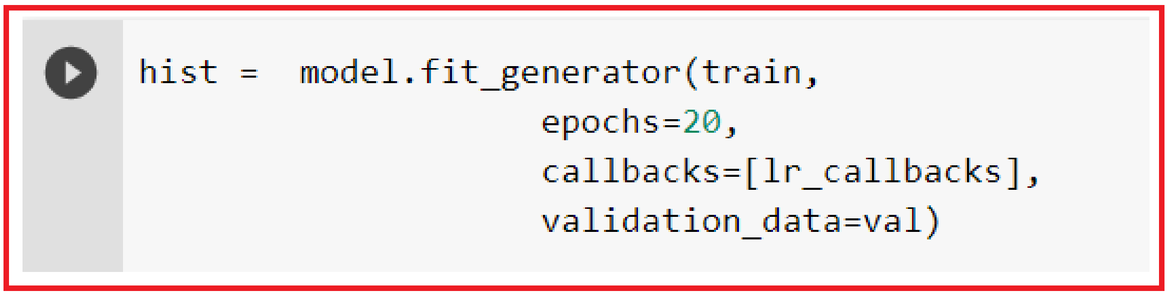
The Figure 8 shows the model building of the MobileNetV2. The activation function is sigmoid and relu. BatchNormalization layer is applied to normalize the data. The Dropout and dense layer is there with 128 units. The code compilation is done using loss function which is sparse categorical cross-entropy and the optimizier is Adms optimizier.



```
[ ]  def scheduler(epoch):
         if epoch <= 2:
             return 0.001
         elif epoch > 2 and epoch <= 15:
             return 0.0001
         else:
             return 0.00001

     lr_callbacks = tf.keras.callbacks.LearningRateScheduler(scheduler)
```

Figure 9: The loss callback scheduler

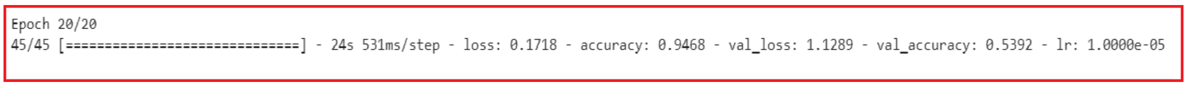
The Figure 9 shows the decalaration of the learning scheduler of the tensorflow keras.



```
hist =  model.fit_generator(train,
                            epochs=20,
                            callbacks=[lr_callbacks],
                            validation_data=val)
```

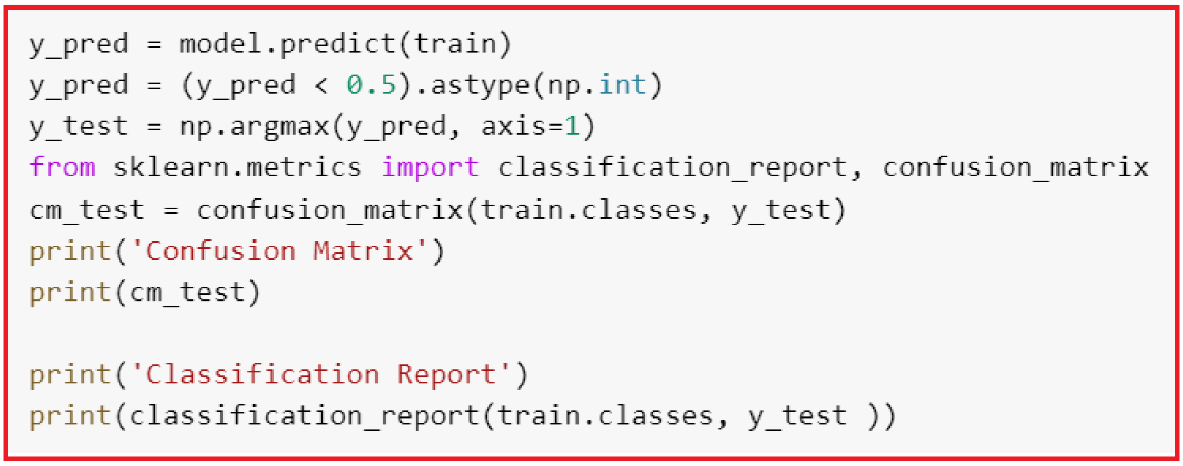
Figure 10: Training and Validation Model 1

The Figure 10 shows the epochs are set to 20 and above is the model training code of the MobileNetV2. Once the Model trained for each epoch it will give the classification report and accuracy which is 94% and loss is 17%.



```
Epoch 20/20
45/45 [==============================] - 24s 531ms/step - loss: 0.1718 - accuracy: 0.9468 - val_loss: 1.1289 - val_accuracy: 0.5392 - lr: 1.0000e-05
```

Figure 11: Accuracy and Loss of Model 1



```
y_pred = model.predict(train)
y_pred = (y_pred < 0.5).astype(np.int)
y_test = np.argmax(y_pred, axis=1)
from sklearn.metrics import classification_report, confusion_matrix
cm_test = confusion_matrix(train.classes, y_test)
print('Confusion Matrix')
print(cm_test)

print('Classification Report')
print(classification_report(train.classes, y_test ))
```

Figure 12: Code to generate the classification Report

The Figure 12 shows the code to generate the classification report and the confusion matrix for our model.

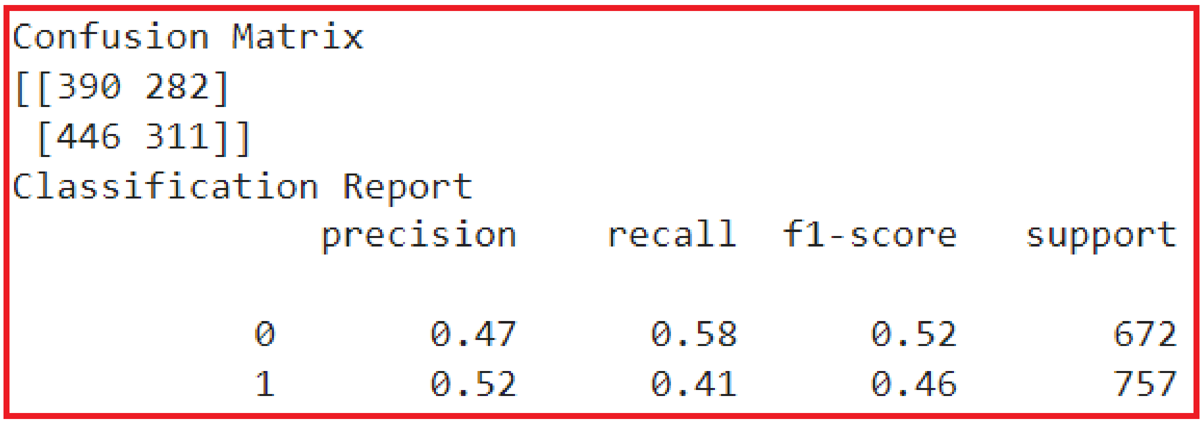

```
Confusion Matrix
[[390 282]
 [446 311]]
Classification Report
              precision    recall  f1-score   support

           0       0.47      0.58      0.52       672
           1       0.52      0.41      0.46       757
```

Figure 13: Classification Report

In the Figure 13 the classification report and confusion matrix is generated for MobileNetV2 model which gives the value of precision, recall and F1-score.

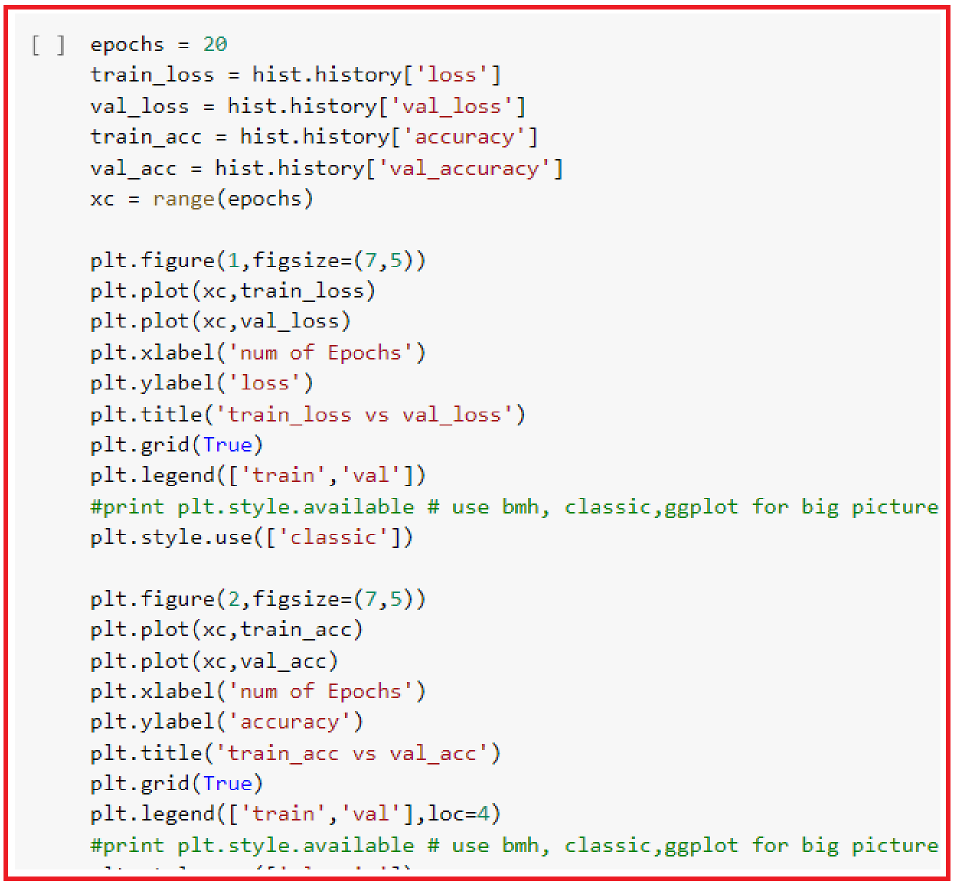

```
[ ] epochs = 20
    train_loss = hist.history['loss']
    val_loss = hist.history['val_loss']
    train_acc = hist.history['accuracy']
    val_acc = hist.history['val_accuracy']
    xc = range(epochs)

    plt.figure(1,figsize=(7,5))
    plt.plot(xc,train_loss)
    plt.plot(xc,val_loss)
    plt.xlabel('num of Epochs')
    plt.ylabel('loss')
    plt.title('train_loss vs val_loss')
    plt.grid(True)
    plt.legend(['train','val'])
    #print plt.style.available # use bmh, classic,ggplot for big picture
    plt.style.use(['classic'])

    plt.figure(2,figsize=(7,5))
    plt.plot(xc,train_acc)
    plt.plot(xc,val_acc)
    plt.xlabel('num of Epochs')
    plt.ylabel('accuracy')
    plt.title('train_acc vs val_acc')
    plt.grid(True)
    plt.legend(['train','val'],loc=4)
    #print plt.style.available # use bmh, classic,ggplot for big picture
```

Figure 14: Code to Plot the Accuracy and Loss Graph

The Figure 14 shows the code for the plotted graphs for the training and validation loss of model.

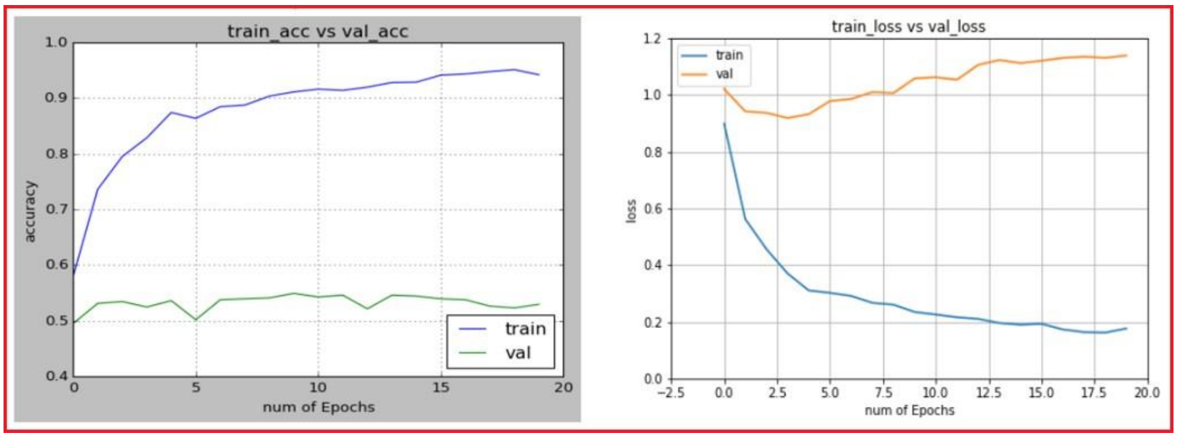
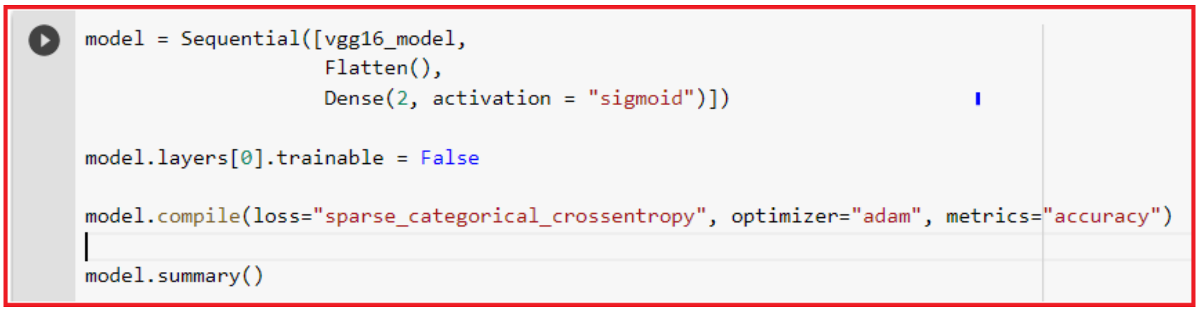


Figure 15: Accuracy and Loss Graph of Training and Validation Data

The Figure 15 shows the graphical representation of the plotted graphs for the training and validation loss of model.

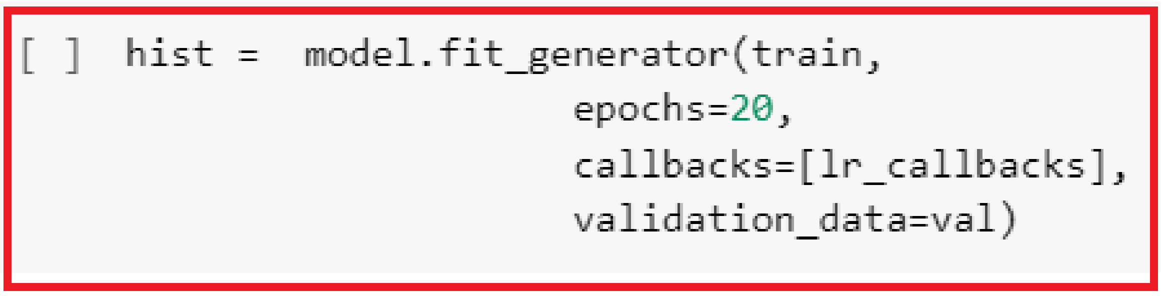
7.2 Implemenatation and Evaluation of Vgg16 (Model 2)



```
model = Sequential([vgg16_model,
                    Flatten(),
                    Dense(2, activation = "sigmoid")])

model.layers[0].trainable = False

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics="accuracy")

model.summary()
```

Figure 16: Building Model 2

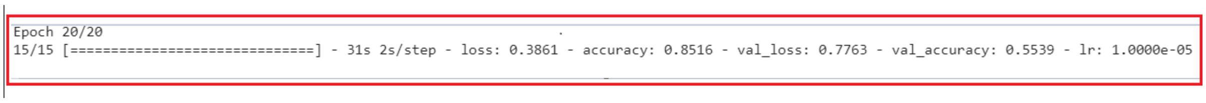
The Figure 16 shows the model building of the Vgg16. The activation function is sigmoid. The last layer is dense layer with activation function sigmoid. The code compilation is done using loss function which is sparse categorical cross-entropy and the optimizier is Adms optimizier.



```
[ ]  hist =  model.fit_generator(train,
                                 epochs=20,
                                 callbacks=[lr_callbacks],
                                 validation_data=val)
```

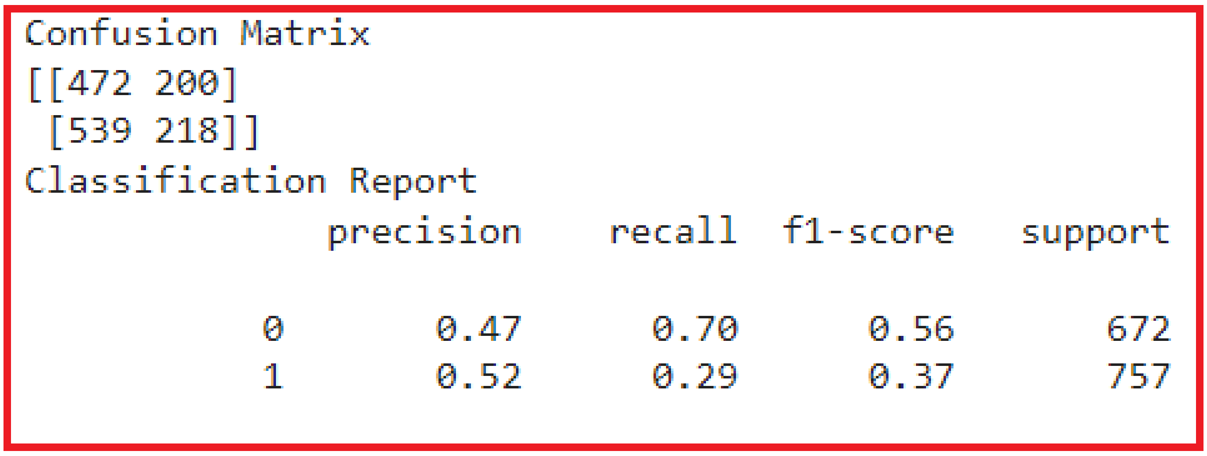
Figure 17: Training and Validation Model 2

The Figure 17 the model training code for the Vgg16 indicates that the epochs are set to 20 and higher. When the model has finished training for each epoch, it will report on classification, accuracy, and loss %.



```
Epoch 20/20
15/15 [==============================] - 31s 2s/step - loss: 0.3861 - accuracy: 0.8516 - val_loss: 0.7763 - val_accuracy: 0.5539 - lr: 1.0000e-05
```

Figure 18: Accuracy and Loss Percentage of Model 2



```
Confusion Matrix
[[472 200]
 [539 218]]
Classification Report
              precision    recall  f1-score   support

           0       0.47      0.70      0.56       672
           1       0.52      0.29      0.37       757
```

Figure 19: Classification Report for the Vgg16 Model

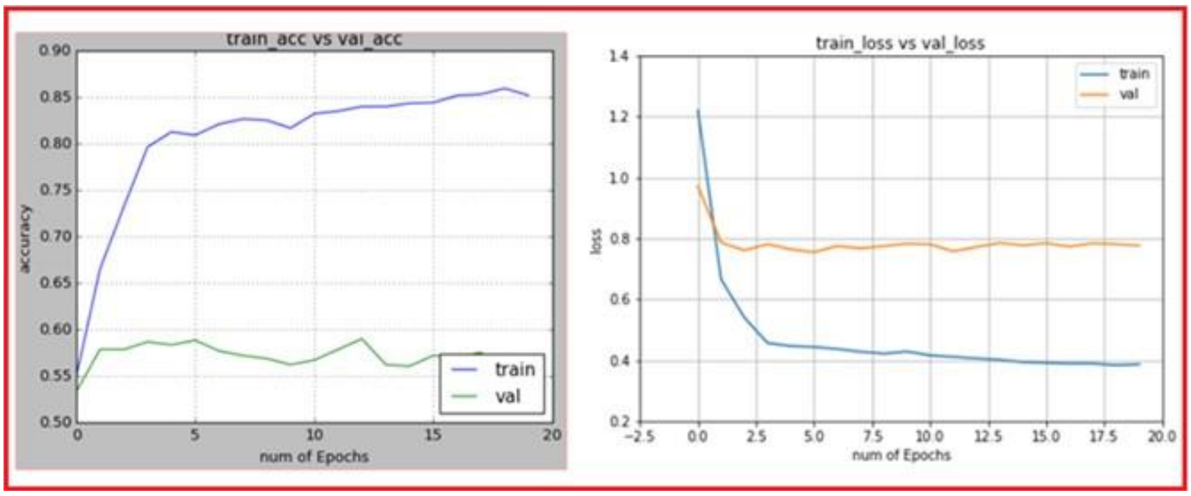
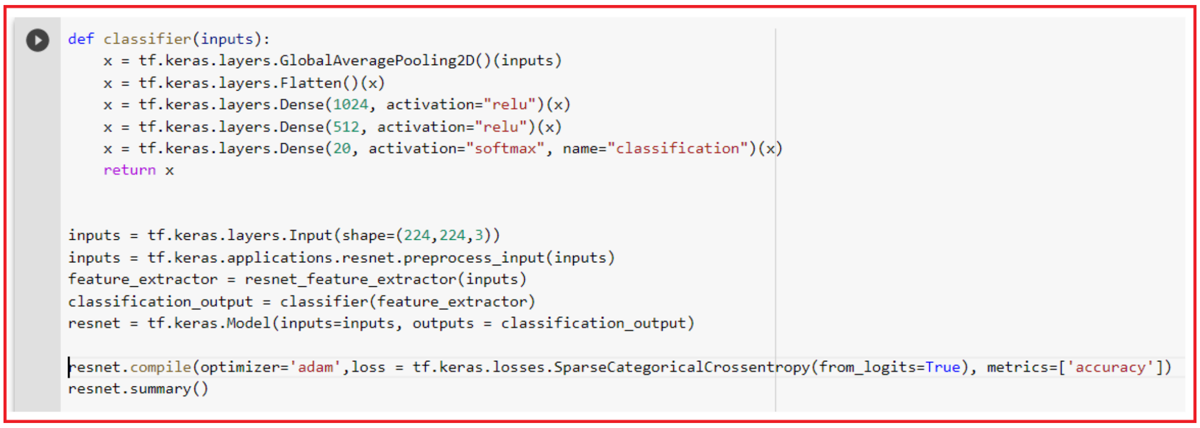


Figure 20: Accuracy and Loss graph of training and validation Data

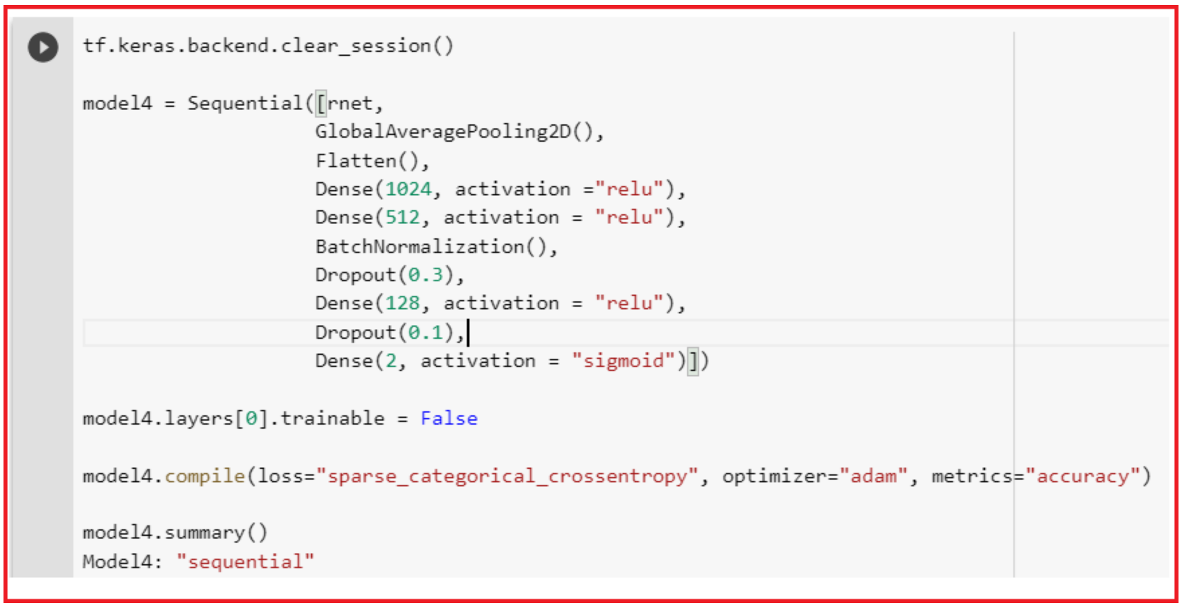
7.3 Implemenatation and Evaluation of ResNet50



```
def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(1024, activation="relu")(x)
    x = tf.keras.layers.Dense(512, activation="relu")(x)
    x = tf.keras.layers.Dense(20, activation="softmax", name="classification")(x)
    return x


inputs = tf.keras.layers.Input(shape=(224,224,3))
inputs = tf.keras.applications.resnet.preprocess_input(inputs)
feature_extractor = resnet_feature_extractor(inputs)
classification_output = classifier(feature_extractor)
resnet = tf.keras.Model(inputs=inputs, outputs = classification_output)

resnet.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
resnet.summary()
```

Figure 21: Added the layer to the Models

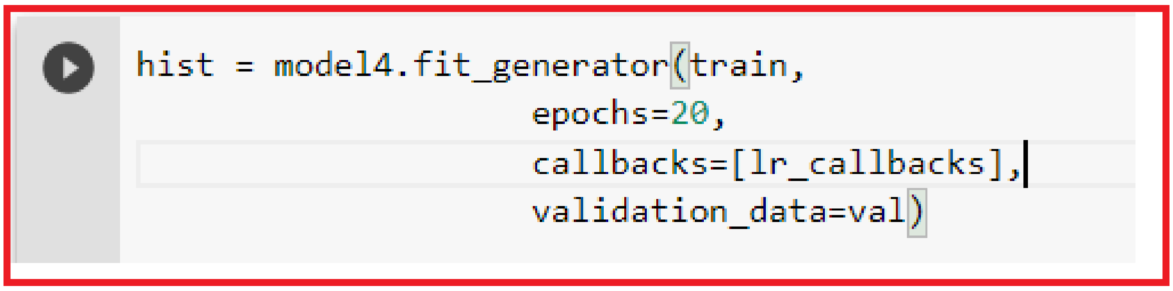
In 21 the custom input layers are are added to the model. The input layers which are added to the model are GlobalAverage polling layer, Dense layer and the sigmoid and relu activation function is used in the model. While compling the model Adm optimizer is used and loss function which is used is sparse cross entropy.



```python
tf.keras.backend.clear_session()

model4 = Sequential([rnet,
                    GlobalAveragePooling2D(),
                    Flatten(),
                    Dense(1024, activation ="relu"),
                    Dense(512, activation = "relu"),
                    BatchNormalization(),
                    Dropout(0.3),
                    Dense(128, activation = "relu"),
                    Dropout(0.1),
                    Dense(2, activation = "sigmoid")])

model4.layers[0].trainable = False

model4.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics="accuracy")

model4.summary()
Model4: "sequential"
```

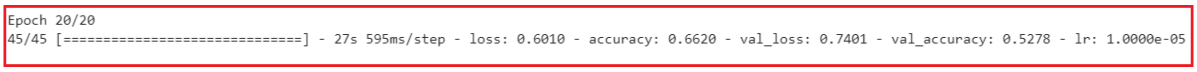
Figure 22: Building Model 3

In 22 the model building of ResNet50 with the global average polling, dense and droupout layer with the loss function is sparse categorical cross-entropy.



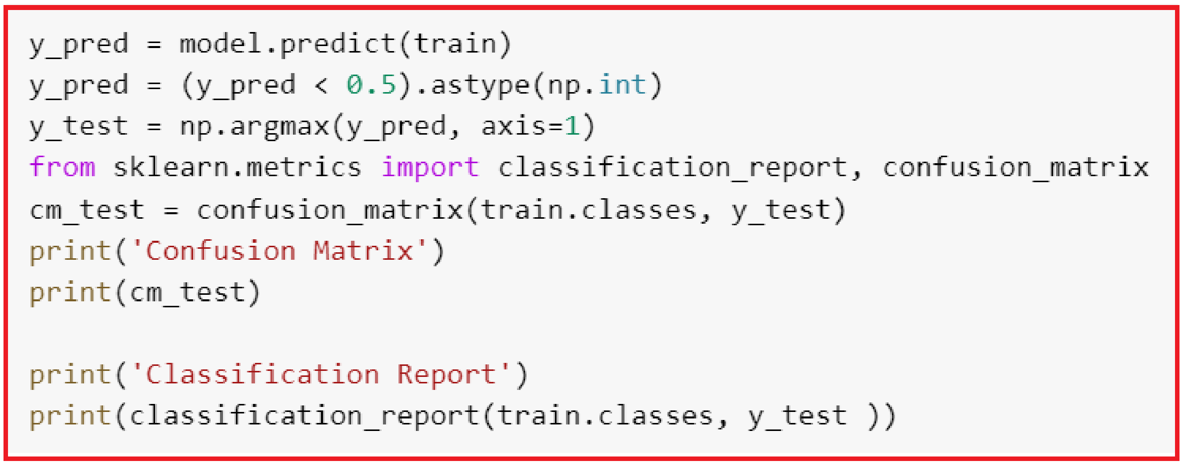
```python
hist = model4.fit_generator(train,
                            epochs=20,
                            callbacks=[lr_callbacks],
                            validation_data=val)
```

Figure 23: Training and Validation of Model 3



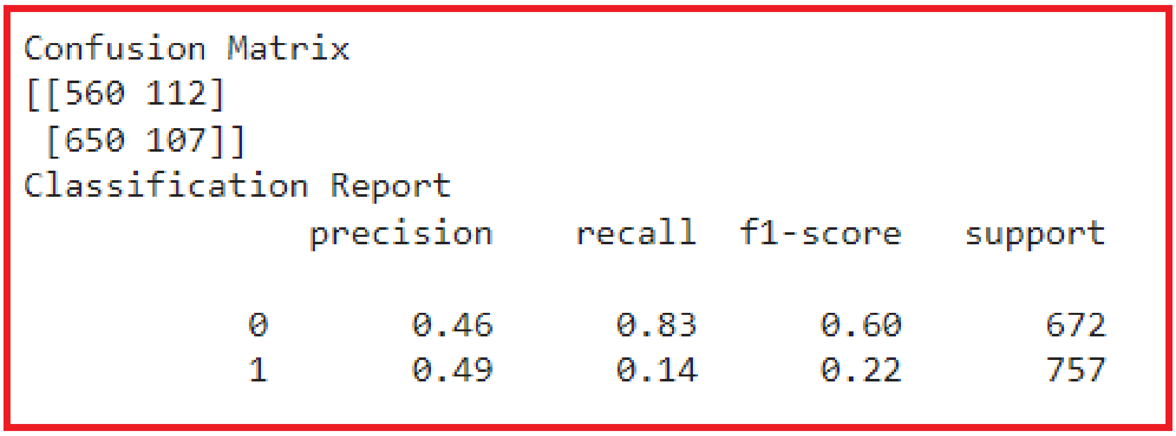
```
Epoch 20/20
45/45 [==============================] - 27s 595ms/step - loss: 0.6010 - accuracy: 0.6620 - val_loss: 0.7401 - val_accuracy: 0.5278 - lr: 1.0000e-05
```

Figure 24: Training and Validation of Model 3



```
y_pred = model.predict(train)
y_pred = (y_pred < 0.5).astype(np.int)
y_test = np.argmax(y_pred, axis=1)
from sklearn.metrics import classification_report, confusion_matrix
cm_test = confusion_matrix(train.classes, y_test)
print('Confusion Matrix')
print(cm_test)

print('Classification Report')
print(classification_report(train.classes, y_test ))
```

Figure 25: Training and Validation of Model 3



```
Confusion Matrix
[[560 112]
 [650 107]]
Classification Report
             precision    recall  f1-score   support

           0       0.46      0.83      0.60       672
           1       0.49      0.14      0.22       757
```

Figure 26: Training and Validation of Model 3

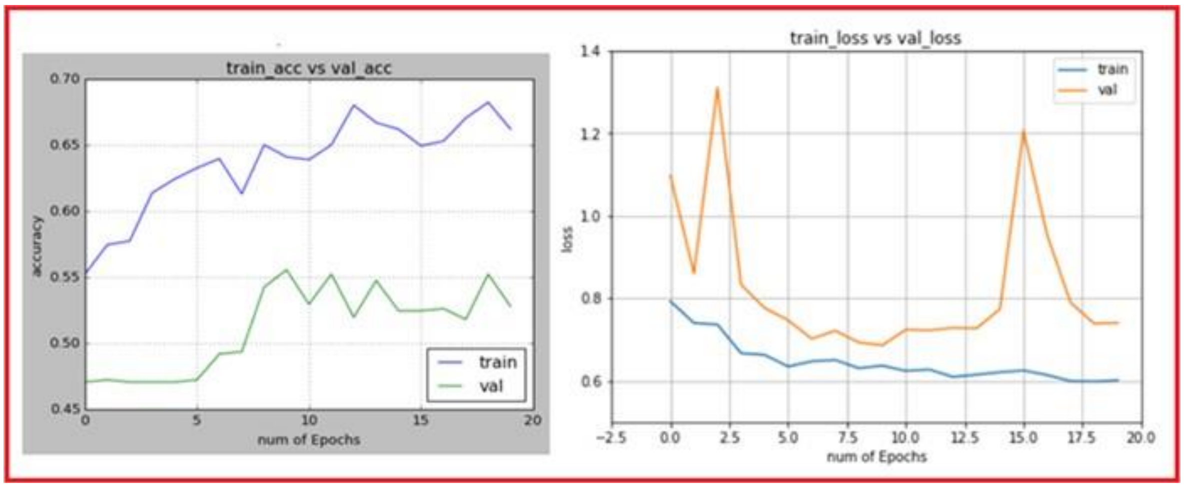


Figure 27: Training and Validation of Model 3

References

Rokhana, R., Herulambang, W. & Indraswari, R. (2021), Multi-class image classification based on mobilenetv2 for detecting the proper use of face mask, in '2021 International Electronics Symposium (IES)', IEEE, pp. 636–641.