

DeepFake Detection using Deep Neural Networks - Configuration Manual

MSc Research Project
Data Analytics

Ambuj Agnihotri
Student ID: x19220073

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Ambuj Agnihotri
Student ID:	x19220073
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	16/08/2021
Project Title:	DeepFake Detection using Deep Neural Networks - Configuration Manual
Word Count:	1032
Page Count:	13

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:	<i>Ambuj Agnihotri</i>
Date:	16th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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 keep a copy on computer.	<input type="checkbox"/>

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):	

DeepFake Detection using Deep Neural Networks - Configuration Manual

Ambuj Agnihotri
x19220073

1 Section 1

The configuration manual contains a full explanation of the environmental setup used during the development of a research project: “DeepFake Detection using Deep Neural Networks”. Hardware and software specifications are covered in Section 2, Section 3 deals with data sources, while section 4 deals with implementation.

2 System Configuration

2.1 Hardware Configuration

Table 1: Hardware Configurations

Features	Versions
Operating System	Windows 10 Home (2019 Microsoft Corporation.)
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
RAM	8GB
System Type	64- bit Operating System, x64-based processor
Hard Drive	1TB

2.2 Software Configuration

The research was carried out using two software environments: 1) Google Colaboratory and 2) Anaconda’s Jupyter notebook for deep learning models written in Python programming language. The Python code is exhibited using Anaconda version 4.10.3 in the Jupyter Notebook (6.0.3) web application platform. Jupyter Notebook uses Python 3.8.3 and Google Colab uses python 3.7.11 as its environment. The following are the important libraries that were utilized during the research project.

- Numpy : 1.19.5
- Keras : 2.6.0
- Matplotlib : 3.2.2

- sklearn : 0.23.1
- Tensorflow (Jupyter Notebook): 2.6.0
- Tensorflow (Google Colab) : 2.5.0

3 Data Sources

Flickr-Faces-High Quality (FFHQ)¹ dataset contains 70,000 high-quality PNG images with a 1024×1024 pixel resolution, representing a wide range of race, age, and image background. Eyeglasses, sunglasses, hats, and other accessories are also adequately covered. For their Style-Based Generator Architecture, which is publicly accessible on GitHub, Karras et al. (2019) had used the FFHQ dataset to create deepfake images which are selected as fake images for this research².

4 Implementation

1000 real and 1000 deepfake images are obtained from the two sources stated above and stored in a folder named dataset, based on the available computing capacity.

4.1 Data Pre-processing

After Dataset collection, we used splitfolders library from python to split the dataset into training, validation and testing sets in ratio of 80%, 10% and 10% respectively as shown in figure 1. It is saved into a new folder named 'split_dataset'. This folder is also uploaded on Google drive to be used in Google Colaboratory Code execution as well.

```
In [3]: import splitfolders

In [4]: dataset_path = 'D:/DATA ANALYTICS/Semester3/Research Project/Dataset'
```

```
In [5]: splitfolders.ratio(dataset_path, output='split_dataset', seed=1377, ratio=(.8, .1, .1)) # default values
print('Train/ Val/ Test Split Done!')
```

Copying files: 2000 files [01:01, 32.75 files/s]

Train/ Val/ Test Split Done!

Figure 1: Splitting Dataset

¹<https://github.com/NVlabs/ffhq-dataset>

²<https://github.com/NVlabs/stylegan>

Figure 2 shows importing of all necessary libraries which will be used in code implementation. we have checked the Tensorflow version as well.

```
In [1]: import numpy as np
import os
import tensorflow as tf
from tensorflow.keras import backend as K

tmp_debug_path = '.\\tmp_debug'
print('Creating Directory: ' + tmp_debug_path)
os.makedirs(tmp_debug_path, exist_ok=True)

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras import applications
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Flatten, TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
print('TensorFlow version: ', tf.__version__)

Creating Directory: .\\tmp_debug
TensorFlow version: 2.6.0
```

Figure 2: Importing Libraries

Figure 3 indicates the directory settings of Train, Validation, and Test sets. Each directory path has two folders inside it i.e real and fake which have an equal amount of real and fake images in them which shows the dataset is balanced.

Splitting Dataset into train, validation and test sets

```
3]: dataset_path = 'C:/Users/AMBUJ/OneDrive/Desktop/split_dataset'

train_path = os.path.join(dataset_path, 'train')
val_path = os.path.join(dataset_path, 'val')
test_path = os.path.join(dataset_path, 'test')
```

Figure 3: Path Directory

Some preliminary data visualization is done using matplotlib python library. Figure 4 depicts that train set real and fake images is plotted using load image function and output 1024*1024 resolution images are obtained.

```
def plot_img(path, set_):
    dir_ = os.path.join(path, 'train', set_)
    k = 0
    fig, ax = plt.subplots(3,3, figsize=(8,8))
    fig.suptitle(set_ + 'Faces')
    for j in range(3):
        for i in range(3):
            img = load_img(os.path.join(dir_, os.listdir(os.path.join(dir_))[k]))
            ax[j,i].imshow(img)
            ax[j,i].set_title("")
            ax[j,i].axis('off')
            k +=1
    plt.suptitle(set_ + ' Faces')
    return plt

plot_img(dataset_path, 'real').show()
```

Figure 4: Preliminary Data Visualisation

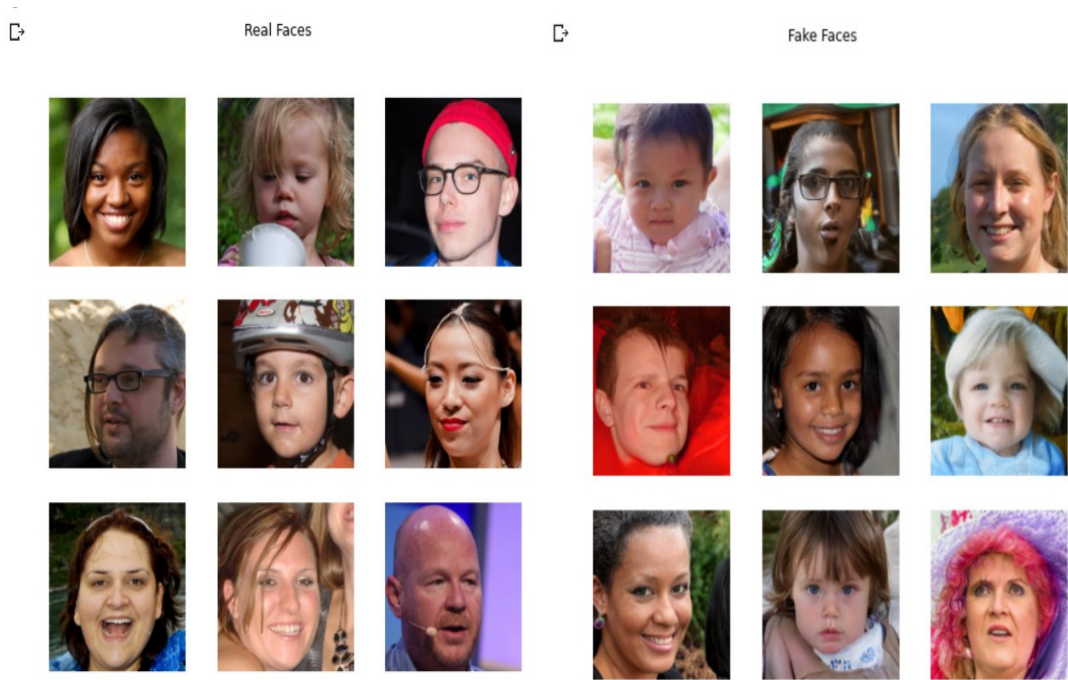


Figure 5: Fake and Real Images

4.2 Data Transformation

Data Augmentation is implemented above on a training set with different parameters such as rotation range as 10, horizontal flip as True, zoom range as 0.1, etc. Resizing of the input image as per computation capacity is been set to $350 * 350$ and batch size is chosen as 16. Below is the code for the same.


```

input_size = 350
batch_size_num = 16
train_datagen = ImageDataGenerator(
    rescale = 1/255,    #rescale the tensor values to [0,1]
    rotation_range = 10,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.2,
    zoom_range = 0.1,
    horizontal_flip = True,
    fill_mode = 'nearest'
)

train_generator = train_datagen.flow_from_directory(
    directory = train_path,
    target_size = (input_size, input_size),
    color_mode = "rgb",
    class_mode = "binary",    #"categorical", "binary", "sparse", "input"
    batch_size = batch_size_num,
    shuffle = True
    #save_to_dir = tmp_debug_path
)

val_datagen = ImageDataGenerator(
    rescale = 1/255    #rescale the tensor values to [0,1]
)

val_generator = val_datagen.flow_from_directory(
    directory = val_path,
    target_size = (input_size, input_size),
    color_mode = "rgb",
    class_mode = "binary",    #"categorical", "binary", "sparse", "input"
    batch_size = batch_size_num,
    shuffle = True
    #save_to_dir = tmp_debug_path
)

test_datagen = ImageDataGenerator(
    rescale = 1/255    #rescale the tensor values to [0,1]
)

test_generator = test_datagen.flow_from_directory(
    directory = test_path,
    classes=['real', 'fake'],
    target_size = (input_size, input_size),
    color_mode = "rgb",
    class_mode = None,
    batch_size = 1,
    shuffle = False
)

Found 1600 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Found 200 images belonging to 2 classes.

```

Figure 6: Data Augmentation

```

bs = 1
x, y = 350, 350
train_datagen_plot = ImageDataGenerator(
    featurewise_center=True,
    rotation_range = 10,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.2,
    zoom_range = 0.1,
    horizontal_flip = True,
    fill_mode = 'nearest'
)
training_set_plot = train_datagen_plot.flow_from_directory(dataset_path + '/train',
    class_mode='binary',
    shuffle=True,
    target_size=(x,y),
    batch_size=bs
)

# generate samples and plot
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

# generate batch of images
for i in range(3):
    # convert to unsigned integers
    image = next(training_set_plot)[0].astype('uint8')
    image1 = image.reshape(350,350,3)
    # plot image
    ax[i].imshow(image1)
    ax[i].axis('on')

```

Figure 7: Post Augmentation Visualisation

After Augmentation, Code for Data visualisation executed again to confirm changes of resizing and re-scaling of images at 350 * 350.

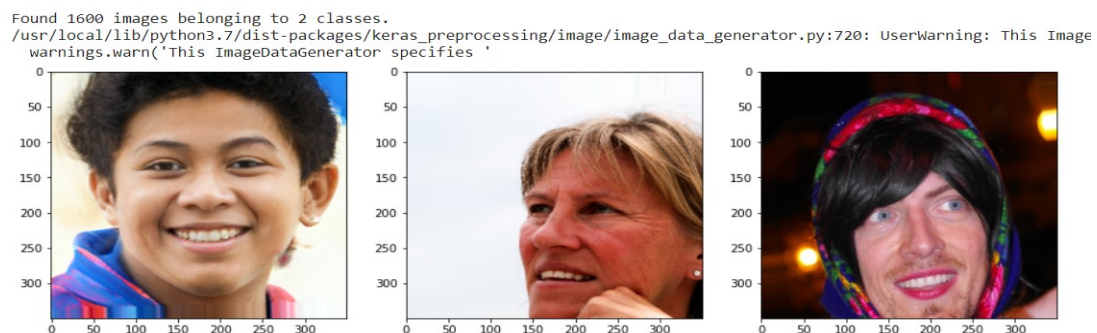


Figure 8: Images After Resizing

4.3 Model Implementation and Evaluation

4.3.1 EfficientNetB4-LSTM (Model 1)

```
n_outputs = 1 #Binary Classification
# Defining a EfficientNetB4 Architecture
efficient_net = EfficientNetB4(
    weights = 'imagenet',
    input_shape = (input_size, input_size, 3),
    include_top = False
)
## define model
model = Sequential()
#### CNN Efficient B4 Layer
model.add(efficient_net)
model.add(Dense(units = 512, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 128, activation = 'relu'))
model.add(TimeDistributed(Flatten()))
#### Adding LSTM Layer to Model
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='sigmoid'))
#### Print Model Summary
model.summary()
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model1.png')
```

Figure 9: Model 1 Building

Above code shows hybrid model building containing EfficientNetB4 and LSTM. n_output parameter is set to be 1 for binary classification (i.e. fake or real). Pretrained CNN on weights as Imagenet is used with LSTM model. Activation function for Dense layer with n_outputs is chosen as sigmoid.

Code for model compilation and saving model checkpoints while training can be seen in figure 6. Learning rate of 0.0001, loss as binary cross-entropy, and metrics as accuracy are chosen while compiling code. Early stopping feature on Validation loss is implemented with the patience of 5 epochs. the best model is saved with the Model checkpoint feature based on validation loss for testing on the test set.

```
model.compile(optimizer = Adam(lr=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
checkpoint_filepath = './tmp_checkpoint'
print('Creating Directory: ' + checkpoint_filepath)
os.makedirs(checkpoint_filepath, exist_ok=True)

custom_callbacks = [
    EarlyStopping(
        monitor = 'val_loss',
        mode = 'min',
        patience = 5,
        verbose = 1
    ),
    ModelCheckpoint(
        filepath = os.path.join(checkpoint_filepath, 'best_model.h5'),
        monitor = 'val_loss',
        mode = 'min',
        verbose = 1,
        save_best_only = True
    )
]
```

Figure 10: Code Compilation Model (1)

Setting the epochs at 20, the model training code can be seen below. After training each epoch, the model automatically validates the model on the validation set which gives us validation loss and accuracy for each epoch.

Training and Validating Model


```
 # Train Model
num_epochs = 20
history = model.fit_generator(
    train_generator,
    epochs = num_epochs,
    steps_per_epoch = len(train_generator),
    validation_data = val_generator,
    validation_steps = len(val_generator),
    callbacks = custom_callbacks
)
print(history.history)
```

Figure 11: Training and Validation Model (1)

The Below code is for training and validation accuracies and losses are plotted after training to get a clear idea of model training.

```
# Plot results
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label = 'Training Accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label = 'Training loss')
plt.plot(epochs, val_loss, 'b', label = 'Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

Figure 12: Training and Validation Plots Model (1)

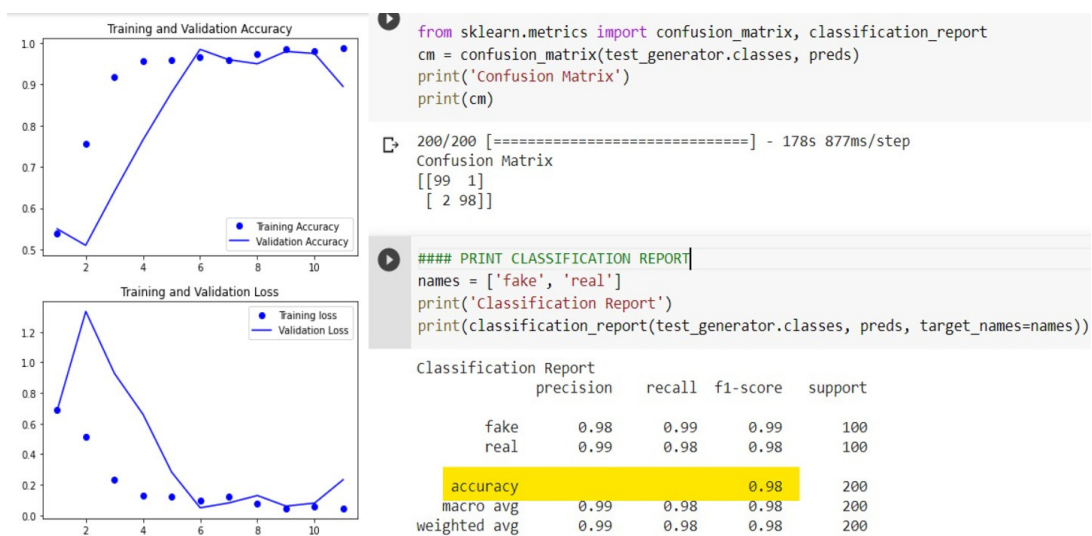


Figure 13: Plots and Accuracy of Model (1)

Results can be seen below for the plots. All points and line are nearby except for epochs except for epoch 2. Figure 13 and 14 shows classification report and code respectively of testing of best model obtained from training on test sets and finally making predictions and classification report is printed to get the performance of the model using accuracy, F-1 score. Training and test Accuracy of the model is 98.75% and 98% respectively.

Testing on test set with best trained model from above

```
# load the saved model that is considered the best
best_model = load_model(os.path.join(checkpoint_filepath, 'best_model.h5'))

# Generate predictions
test_generator.reset()

preds = best_model.predict(
    test_generator,
    verbose = 1
)
preds = (preds < 0.5).astype(np.int)

from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(test_generator.classes, preds)
print('Confusion Matrix')
print(cm)
```

Figure 14: Code for Testing Best Model (1)

4.3.2 InceptionV3-LSTM (Model 2)

Firstly, we imported cnn architecture of inceptionv3 from keras.applications.inception library which is shown in figure below. Inception has other versions as well but needed v3 to be implemented for our research.

```
import tensorflow as tf
from tensorflow.keras import backend as K
import numpy as np
import os
tmp_debug_path = '.\\tmp_debug'
print('Creating Directory: ' + tmp_debug_path)
os.makedirs(tmp_debug_path, exist_ok=True)

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras import applications
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Flatten, TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
print('TensorFlow version: ', tf.__version__)
```

Figure 15: Importing InceptionV3 Model

Preprocessing steps kept same as input for this model is also batch size 16 and image size 350*350. Only difference is that instead of using EfficientNetB4, InceptionV3 CNN architecture is used with LSTM. During Model building that step can be seen clearly from image below. Rest of code remains the same.


```

n_outputs = 1 #Binary Classification
inception_model = InceptionV3(input_shape = (input_size, input_size, 3), include_top = False, weights = 'imagenet')
## Hybrid Model
model = Sequential()
model.add(inception_model)
model.add(Dense(units = 512, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 128, activation = 'relu'))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='sigmoid'))

```

Figure 16: InceptionV3-LSTM Model Building (Model 2)

Training and validation accuracy of InceptionV3-LSTM model is 98.37% and 96% respectively. Plots obtained and classification report can be seen from below figure.

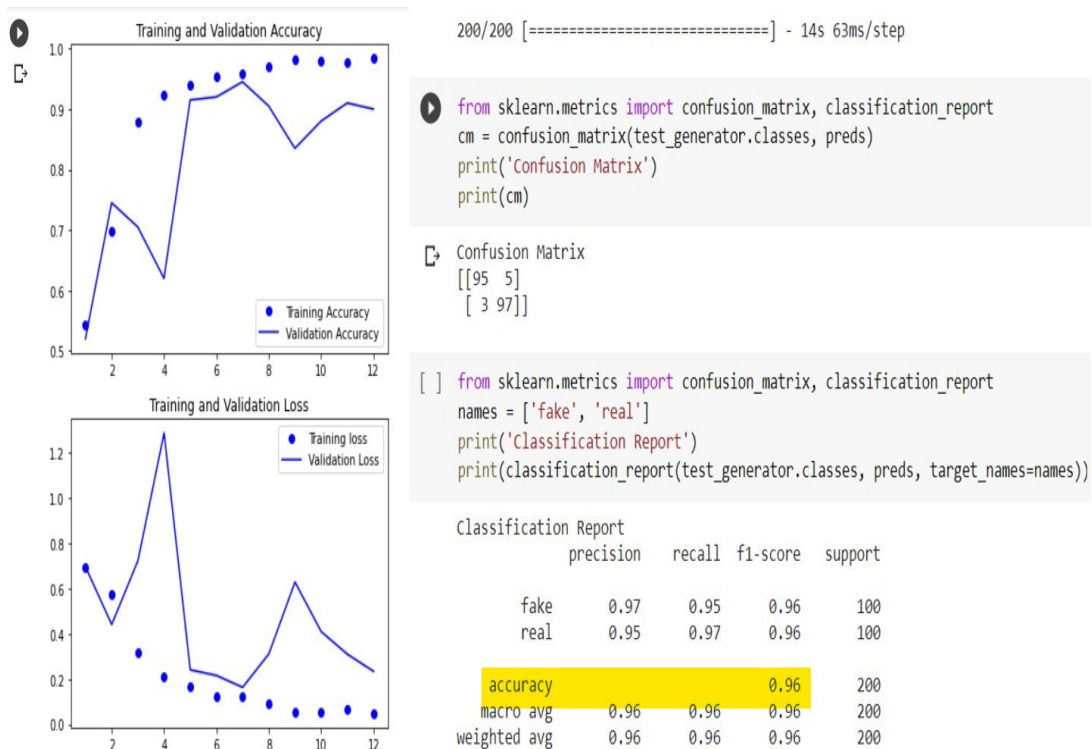


Figure 17: Classification Report of Model 2

4.3.3 InceptionResNetv2-LSTM (Model 3)

Firstly, we imported cnn architecture of InceptionResNetV2 from keras.applications.inception_resnet_v2 library which is shown in figure below. Inceptionresnet has other versions as well but v2 is needed to be implemented for our research.

```
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras import backend as K

tmp_debug_path = '.\\tmp_debug'
print('Creating Directory: ' + tmp_debug_path)
os.makedirs(tmp_debug_path, exist_ok=True)

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras import applications
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Flatten, TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt
print('TensorFlow version: ', tf.__version__)

Creating Directory: .\\tmp_debug
TensorFlow version: 2.6.0
```

Figure 18: Importing InceptionResNetV2 Model

```
n_outputs = 1 #Binary Classification
inceptionresnetv2 = InceptionResNetV2(input_shape = (input_size, input_size, 3), include_top = False, weights = 'imagenet')

model = Sequential()
model.add(inceptionresnetv2)
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='sigmoid'))

# Compile model
model.compile(optimizer = Adam(lr=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 19: InceptionResNetV2 Model Building (Model 3)

Only difference is CNN architecture which is InceptionResNetv2 in model 3 with LSTM. Rest of things remains same. Due to larger architecture, training time was more for this model and is implemented using Jupyter Notebook. Training and test accuracy

for InceptionResNetv2 model is 98.75% and 97%. Plots obtained and classification report can be seen from below figure.

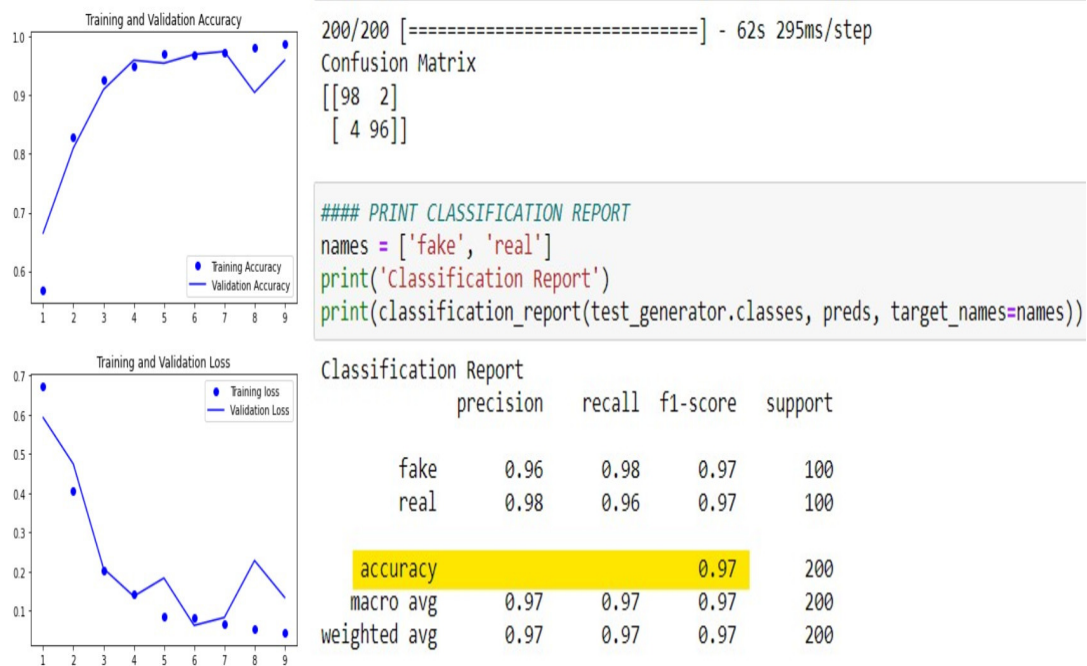


Figure 20: Evaluation And Classification (Model 3)

References

Karras, T., Laine, S. and Aila, T. (2019). A style-based generator architecture for generative adversarial networks.