<u>MINI PROJECT</u>
<u>EC9170 – DEEP LEARNING FOR ELECTRICAL AND COMPUTER ENGINEERS</u>

<u>GROUP MEMBERS:</u>

1. KOKILARAJ.A (2020/E/074)
2. SATHURSHAN.P (2020/E/145)
3. UZAIR U.M. (2020/E/201)

PROJECT TITLE: REAL AND FAKE FACE DETECTION

Introduction

The goal of this project is to develop a deep learning model that classifies images of faces into real and fake categories. The dataset contains two folders named "Real" and "Fake," which include images of real human faces and high-quality photoshopped face images, respectively.

Dataset

The dataset can be accessed through the following link:

https://drive.google.com/file/d/16y2xEwwuf1v_W0BUDv1thzpNu_dnaJZ1/view?usp=sharing

Model Selection

For this project, we propose the following two deep-learning models:

1. Convolutional Neural Network (CNN)
2. Transfer Learning with Pre-trained VGG16 Model

Model Architectures

1. **Convolutional Neural Network (CNN) Architecture**:
   - **Input Layer**: Accepts input images of shape (150, 150, 3).
   - **Convolutional Layers**: Multiple layers with filters of sizes (3x3) to extract features.
   - **Pooling Layers**: Max-pooling layers to reduce dimensionality.
   - **Fully Connected Layers**: Dense layers leading to the final classification layer.
   - **Output Layer**: Sigmoid layer for binary classification (real vs. fake).

2. **Transfer Learning with VGG16 Architecture**:
    - **Base Model**: Pre-trained VGG16 model excluding the top layers.
    - **Custom Top Layers**: Additional dense layers and a sigmoid output layer for binary classification.
    - **Freezing Layers**: Initial layers of VGG16 are frozen to retain pre-trained weights, while later layers are fine-tuned on our dataset.

<u>Evaluation and Comparison</u>

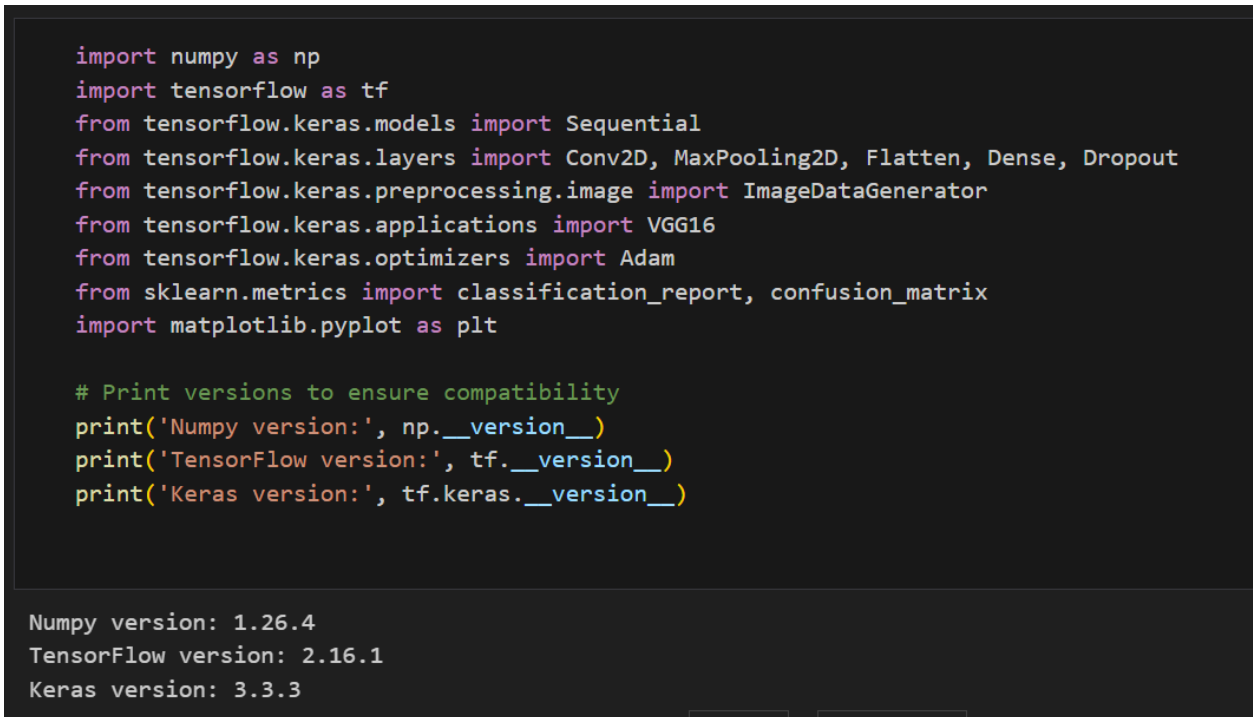
1. **Evaluation Metrics**:

    - CNN Model
        - Achieved validation accuracy of 62%.
        - Loss during validation was 0.6.
        - Classification report and confusion matrix indicated performance on validation data.

    - VGG16 Model
        - Achieved validation accuracy of 0.74%.
        - Loss during validation was 0.7.
        - Classification report and confusion matrix indicated performance on validation data.

2. **Training and Validation accuracy**:
    - Both models were trained for 30 epochs.
    - The training and validation accuracy curves were plotted for comparison.

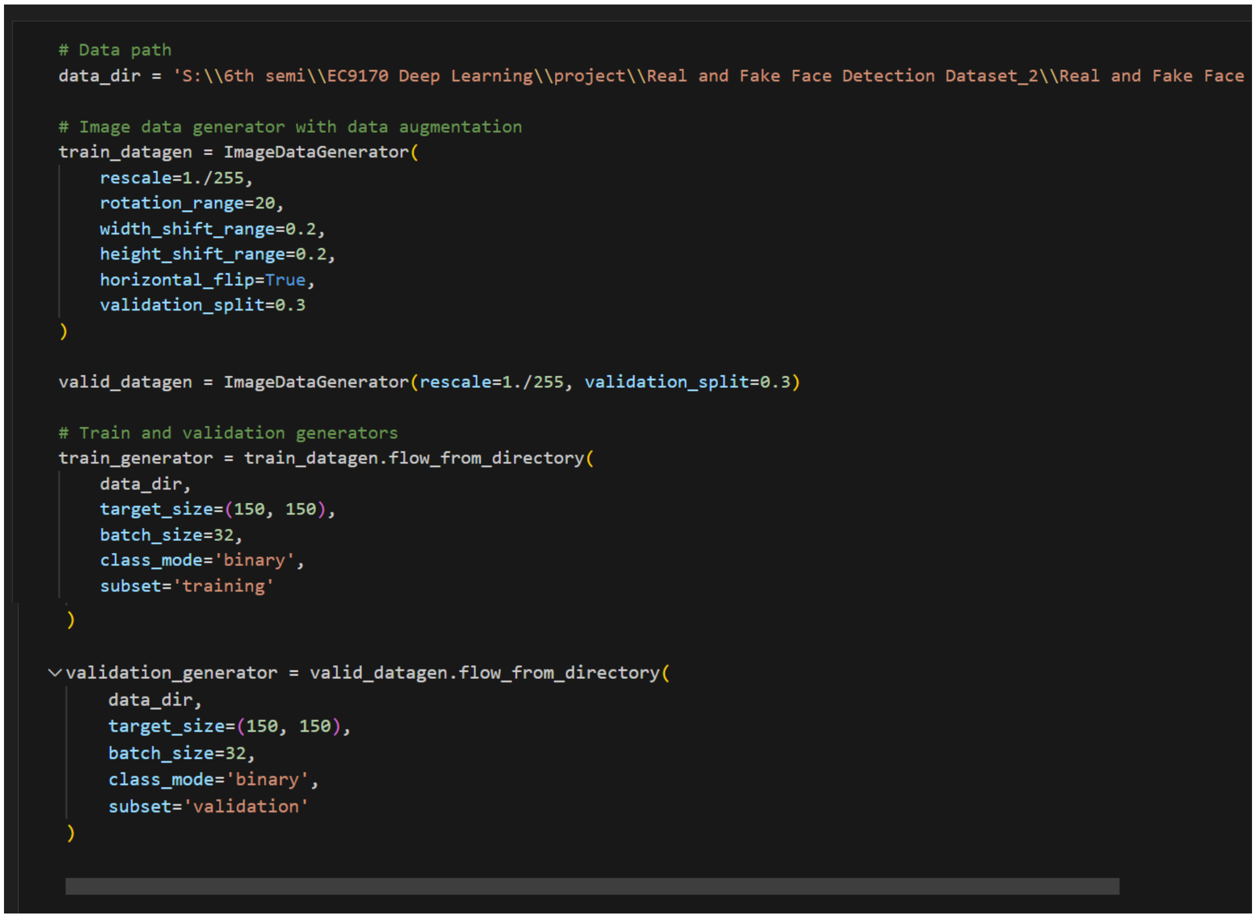
3. **Comparison**:
    - **Accuracy**: VGG16 showed higher accuracy than the CNN model.
    - **Loss**: VGG16 had lower validation loss compared to the CNN model.
    - **Classification Report**: VGG16 demonstrated better precision, recall, and F1-score.
    - **Confusion Matrix**: VGG16 had fewer misclassifications.



```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Print versions to ensure compatibility
print('Numpy version:', np.__version__)
print('TensorFlow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
```

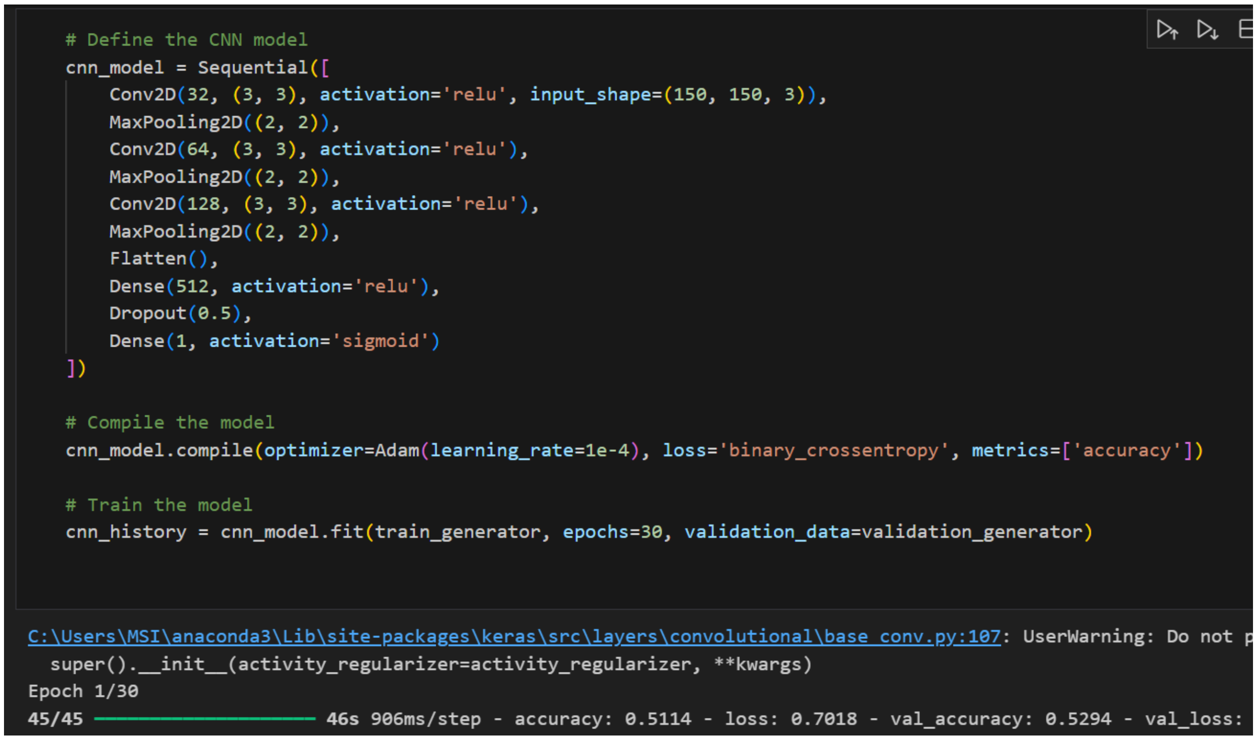
```
Numpy version: 1.26.4
TensorFlow version: 2.16.1
Keras version: 3.3.3
```

Import libraries



```
# Data path
data_dir = 'S:\\6th semi\\EC9170 Deep Learning\\project\\Real and Fake Face Detection Dataset_2\\Real and Fake Face

# Image data generator with data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.3
)

valid_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.3)

# Train and validation generators
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = valid_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

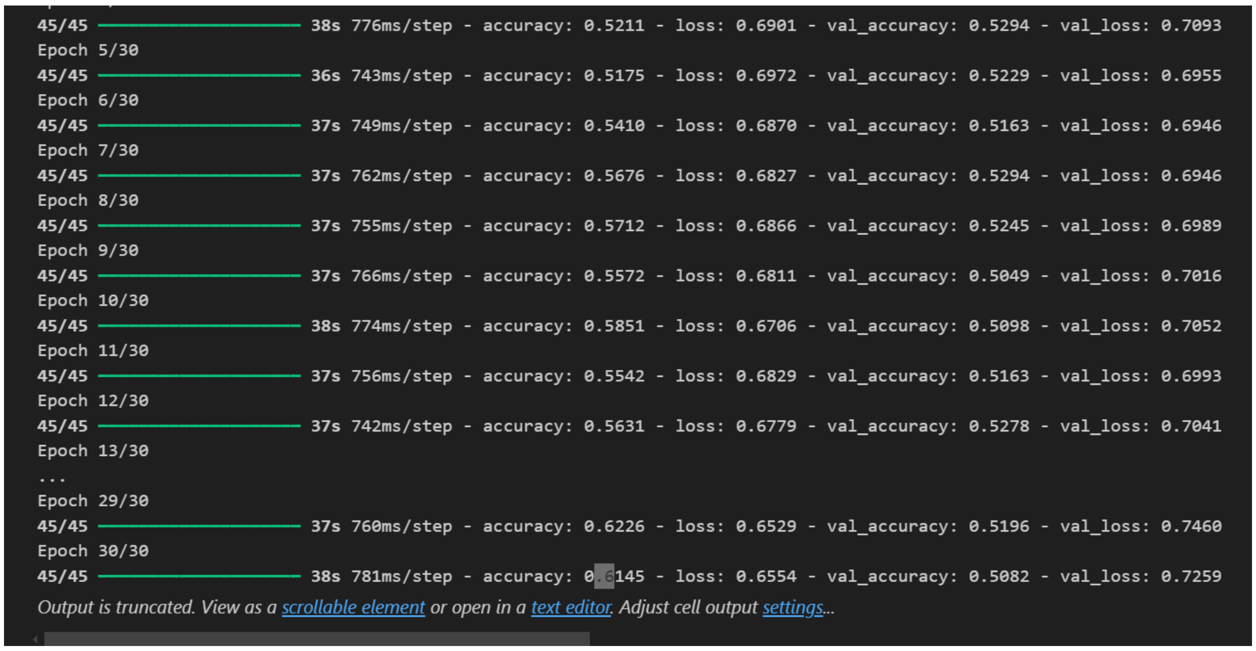
Load the Dataset



```
# Define the CNN model
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
cnn_model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
cnn_history = cnn_model.fit(train_generator, epochs=30, validation_data=validation_generator)
```

```
C:\Users\MSI\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not p
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 46s 906ms/step - accuracy: 0.5114 - loss: 0.7018 - val_accuracy: 0.5294 - val_loss:
```

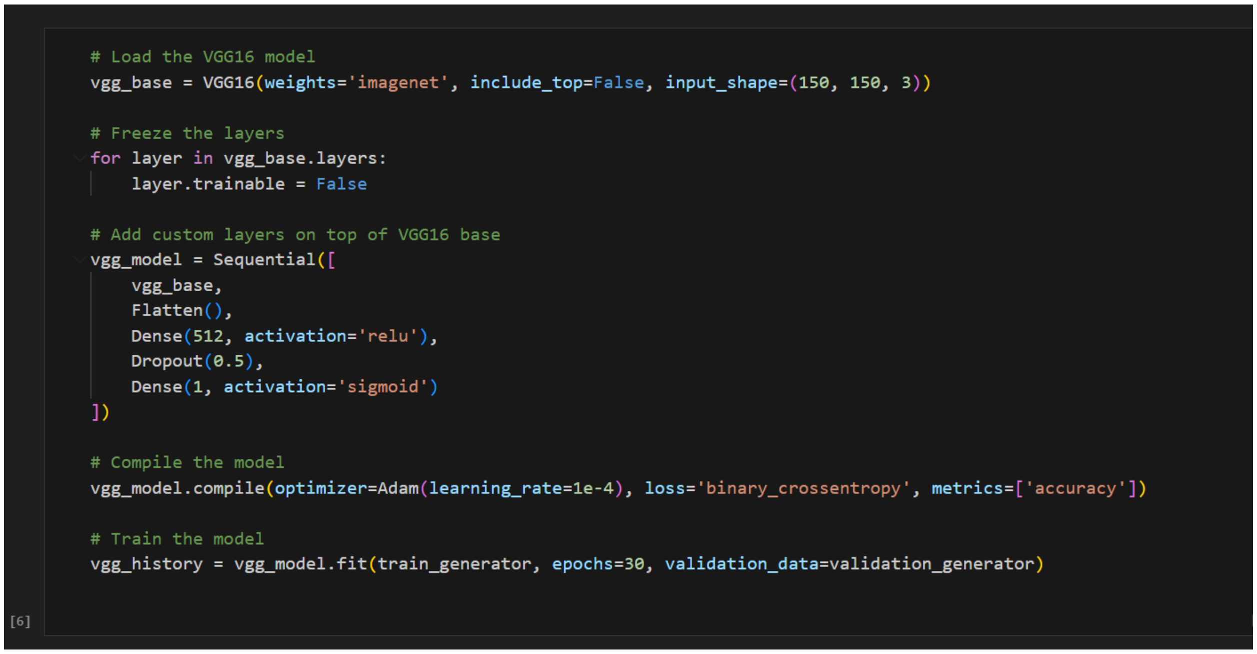
CNN Model



```
45/45 ━━━━━━━━━━━━━━━━━━━━ 38s 776ms/step - accuracy: 0.5211 - loss: 0.6901 - val_accuracy: 0.5294 - val_loss: 0.7093
Epoch 5/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 36s 743ms/step - accuracy: 0.5175 - loss: 0.6972 - val_accuracy: 0.5229 - val_loss: 0.6955
Epoch 6/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 749ms/step - accuracy: 0.5410 - loss: 0.6870 - val_accuracy: 0.5163 - val_loss: 0.6946
Epoch 7/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 762ms/step - accuracy: 0.5676 - loss: 0.6827 - val_accuracy: 0.5294 - val_loss: 0.6946
Epoch 8/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 755ms/step - accuracy: 0.5712 - loss: 0.6866 - val_accuracy: 0.5245 - val_loss: 0.6989
Epoch 9/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 766ms/step - accuracy: 0.5572 - loss: 0.6811 - val_accuracy: 0.5049 - val_loss: 0.7016
Epoch 10/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 38s 774ms/step - accuracy: 0.5851 - loss: 0.6706 - val_accuracy: 0.5098 - val_loss: 0.7052
Epoch 11/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 756ms/step - accuracy: 0.5542 - loss: 0.6829 - val_accuracy: 0.5163 - val_loss: 0.6993
Epoch 12/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 742ms/step - accuracy: 0.5631 - loss: 0.6779 - val_accuracy: 0.5278 - val_loss: 0.7041
Epoch 13/30
...
Epoch 29/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 37s 760ms/step - accuracy: 0.6226 - loss: 0.6529 - val_accuracy: 0.5196 - val_loss: 0.7460
Epoch 30/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 38s 781ms/step - accuracy: 0.6145 - loss: 0.6554 - val_accuracy: 0.5082 - val_loss: 0.7259
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

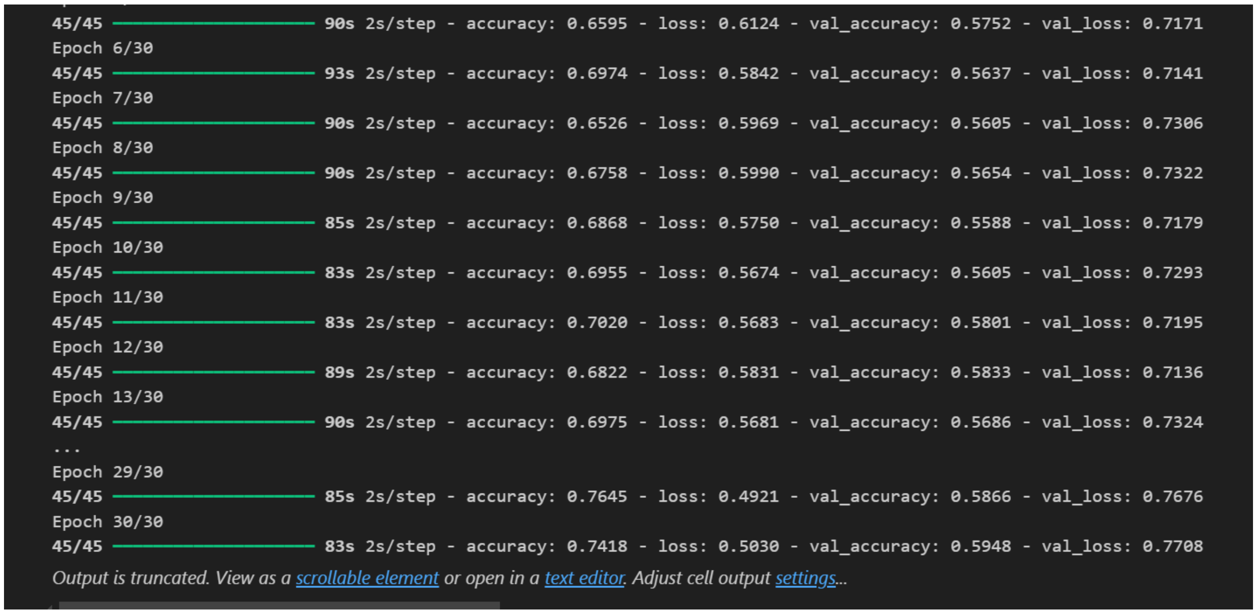
CNN Trained



```python
# Load the VGG16 model
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Freeze the layers
for layer in vgg_base.layers:
    layer.trainable = False

# Add custom layers on top of VGG16 base
vgg_model = Sequential([
    vgg_base,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
vgg_model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
vgg_history = vgg_model.fit(train_generator, epochs=30, validation_data=validation_generator)
```

[6]

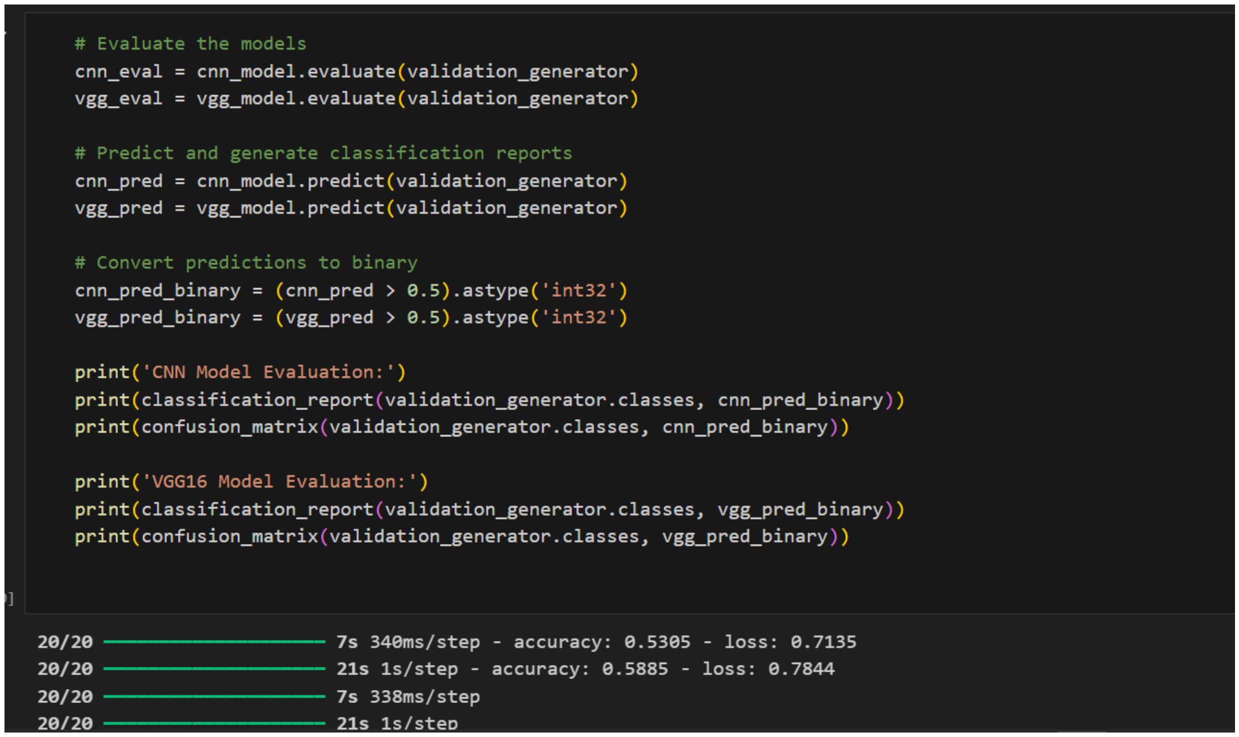
VGG16 Model



```
45/45 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.6595 - loss: 0.6124 - val_accuracy: 0.5752 - val_loss: 0.7171
Epoch 6/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 93s 2s/step - accuracy: 0.6974 - loss: 0.5842 - val_accuracy: 0.5637 - val_loss: 0.7141
Epoch 7/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.6526 - loss: 0.5969 - val_accuracy: 0.5605 - val_loss: 0.7306
Epoch 8/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.6758 - loss: 0.5990 - val_accuracy: 0.5654 - val_loss: 0.7322
Epoch 9/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.6868 - loss: 0.5750 - val_accuracy: 0.5588 - val_loss: 0.7179
Epoch 10/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.6955 - loss: 0.5674 - val_accuracy: 0.5605 - val_loss: 0.7293
Epoch 11/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7020 - loss: 0.5683 - val_accuracy: 0.5801 - val_loss: 0.7195
Epoch 12/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step - accuracy: 0.6822 - loss: 0.5831 - val_accuracy: 0.5833 - val_loss: 0.7136
Epoch 13/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 90s 2s/step - accuracy: 0.6975 - loss: 0.5681 - val_accuracy: 0.5686 - val_loss: 0.7324
...
Epoch 29/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 85s 2s/step - accuracy: 0.7645 - loss: 0.4921 - val_accuracy: 0.5866 - val_loss: 0.7676
Epoch 30/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7418 - loss: 0.5030 - val_accuracy: 0.5948 - val_loss: 0.7708
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

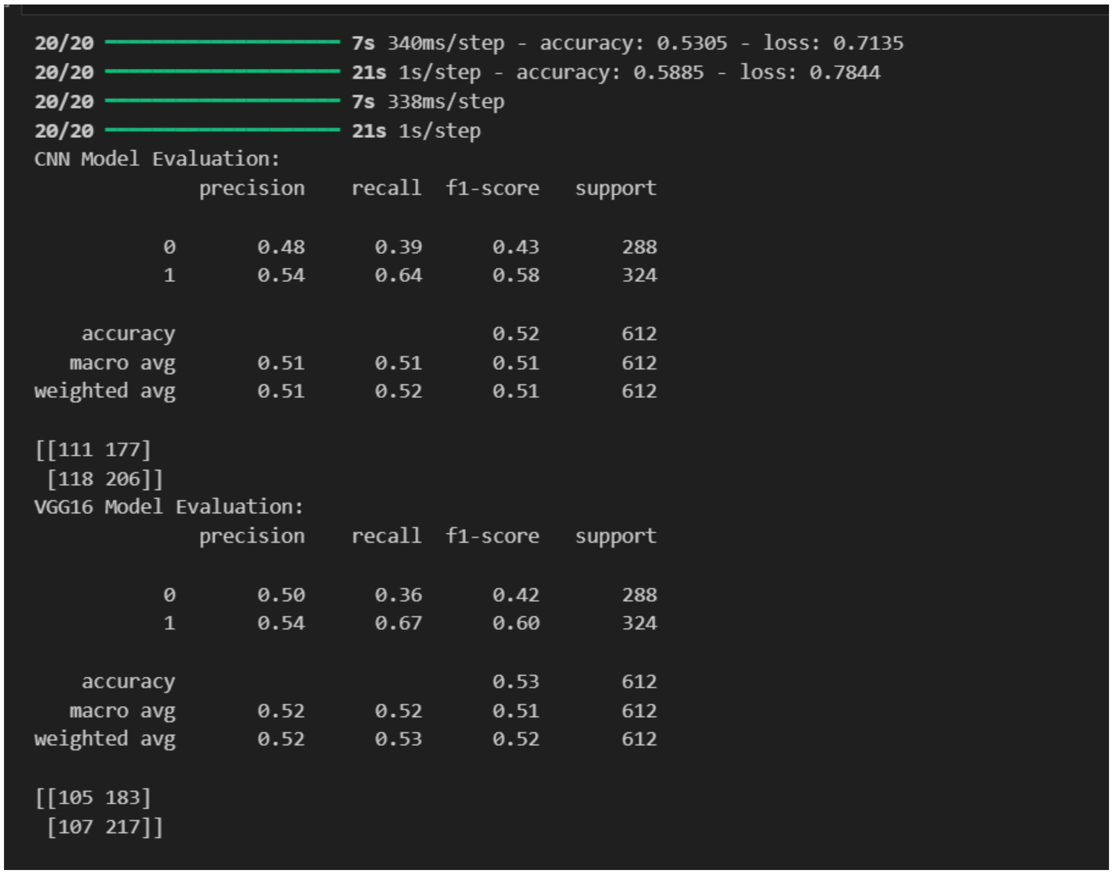
VGG16 Trained



```
# Evaluate the models
cnn_eval = cnn_model.evaluate(validation_generator)
vgg_eval = vgg_model.evaluate(validation_generator)

# Predict and generate classification reports
cnn_pred = cnn_model.predict(validation_generator)
vgg_pred = vgg_model.predict(validation_generator)

# Convert predictions to binary
cnn_pred_binary = (cnn_pred > 0.5).astype('int32')
vgg_pred_binary = (vgg_pred > 0.5).astype('int32')

print('CNN Model Evaluation:')
print(classification_report(validation_generator.classes, cnn_pred_binary))
print(confusion_matrix(validation_generator.classes, cnn_pred_binary))

print('VGG16 Model Evaluation:')
print(classification_report(validation_generator.classes, vgg_pred_binary))
print(confusion_matrix(validation_generator.classes, vgg_pred_binary))
```

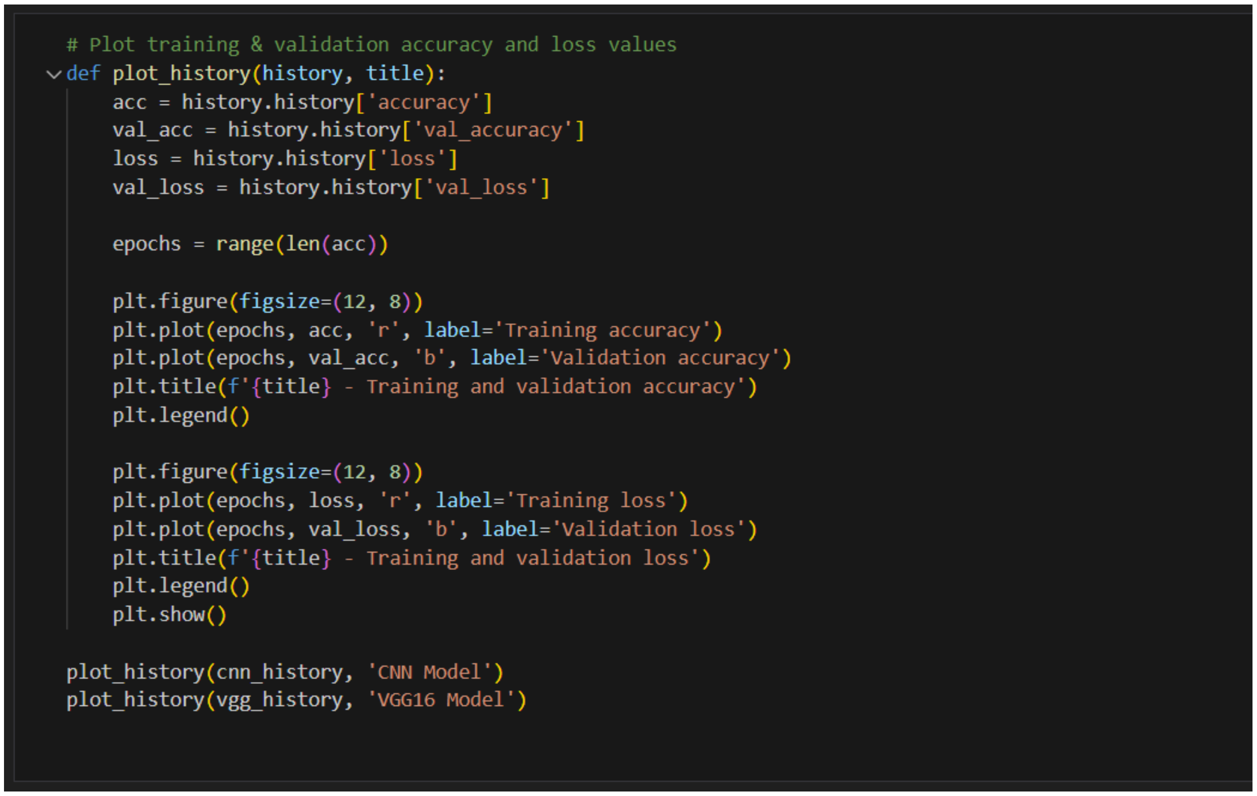
```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 340ms/step - accuracy: 0.5305 - loss: 0.7135
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.5885 - loss: 0.7844
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 338ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step
```

Model Evaluation



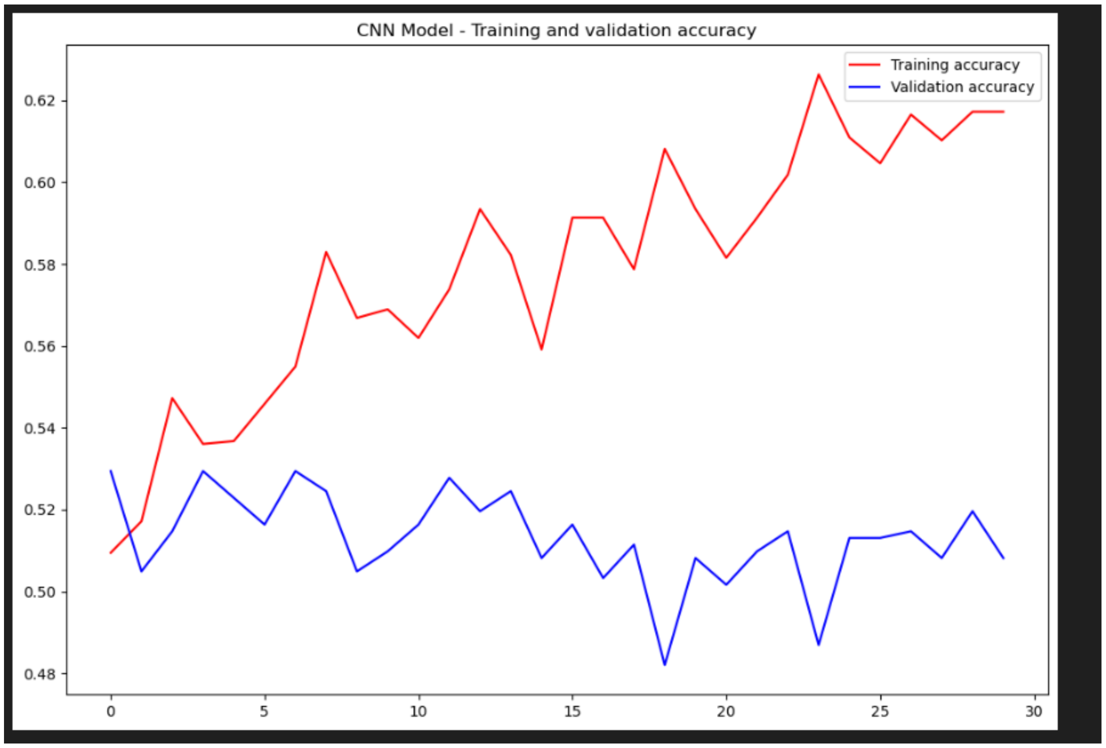
```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 340ms/step - accuracy: 0.5305 - loss: 0.7135
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.5885 - loss: 0.7844
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 338ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step
CNN Model Evaluation:
              precision    recall  f1-score   support

           0       0.48      0.39      0.43       288
           1       0.54      0.64      0.58       324

    accuracy                           0.52       612
   macro avg       0.51      0.51      0.51       612
weighted avg       0.51      0.52      0.51       612

[[111 177]
 [118 206]]
VGG16 Model Evaluation:
              precision    recall  f1-score   support

           0       0.50      0.36      0.42       288
           1       0.54      0.67      0.60       324

    accuracy                           0.53       612
   macro avg       0.52      0.52      0.51       612
weighted avg       0.52      0.53      0.52       612

[[105 183]
 [107 217]]
```

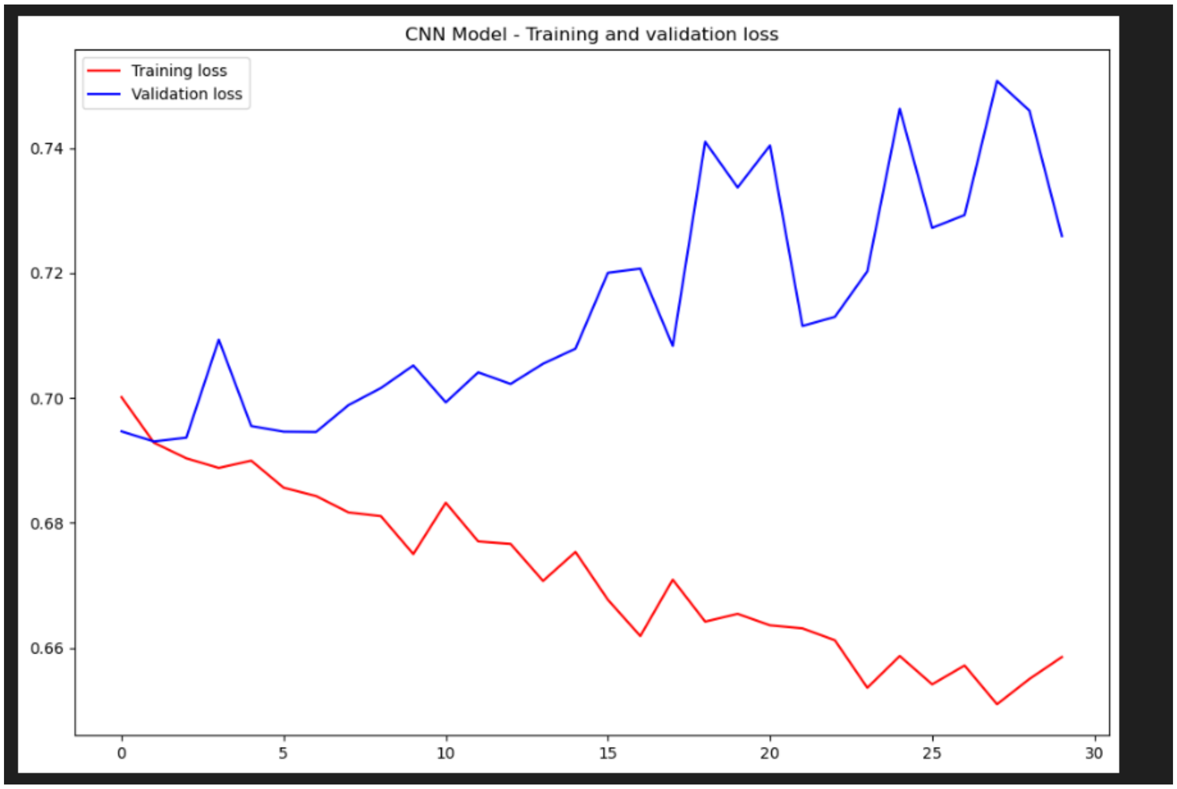
Model Evaluation Output

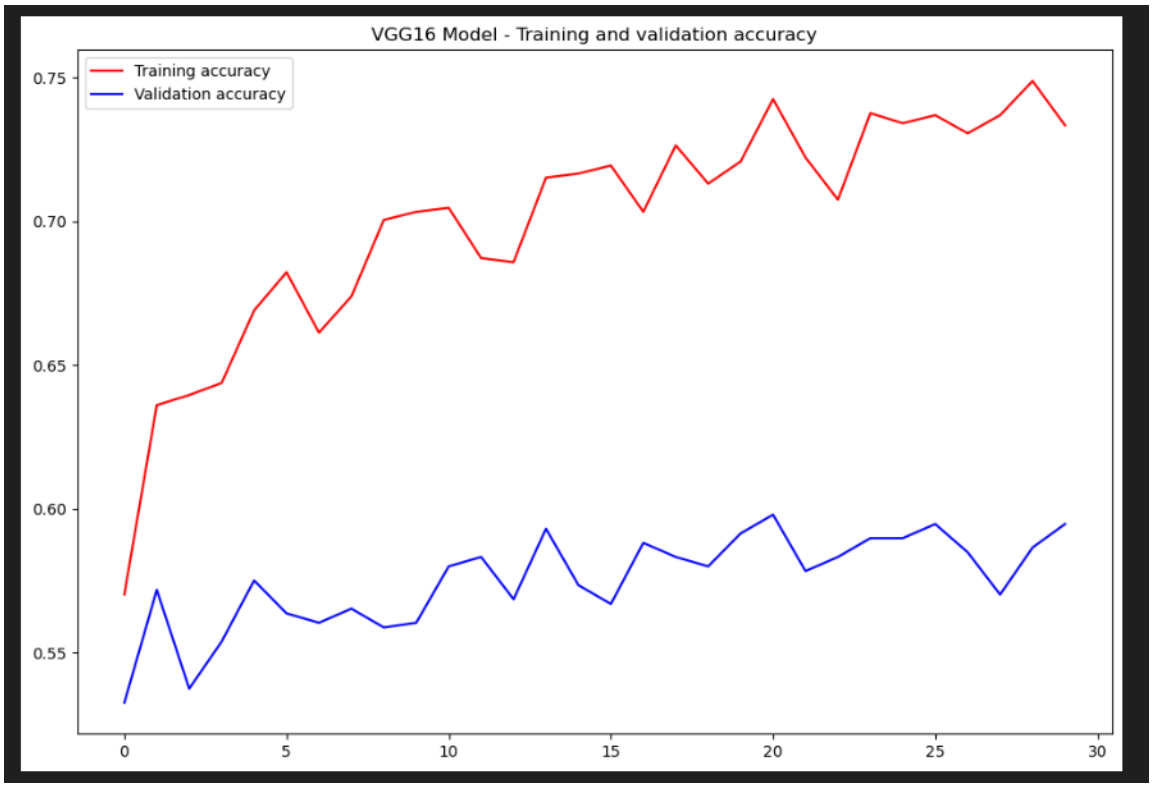

```python
# Plot training & validation accuracy and loss values
def plot_history(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.figure(figsize=(12, 8))
    plt.plot(epochs, acc, 'r', label='Training accuracy')
    plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
    plt.title(f'{title} - Training and validation accuracy')
    plt.legend()

    plt.figure(figsize=(12, 8))
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title(f'{title} - Training and validation loss')
    plt.legend()
    plt.show()

plot_history(cnn_history, 'CNN Model')
plot_history(vgg_history, 'VGG16 Model')
```

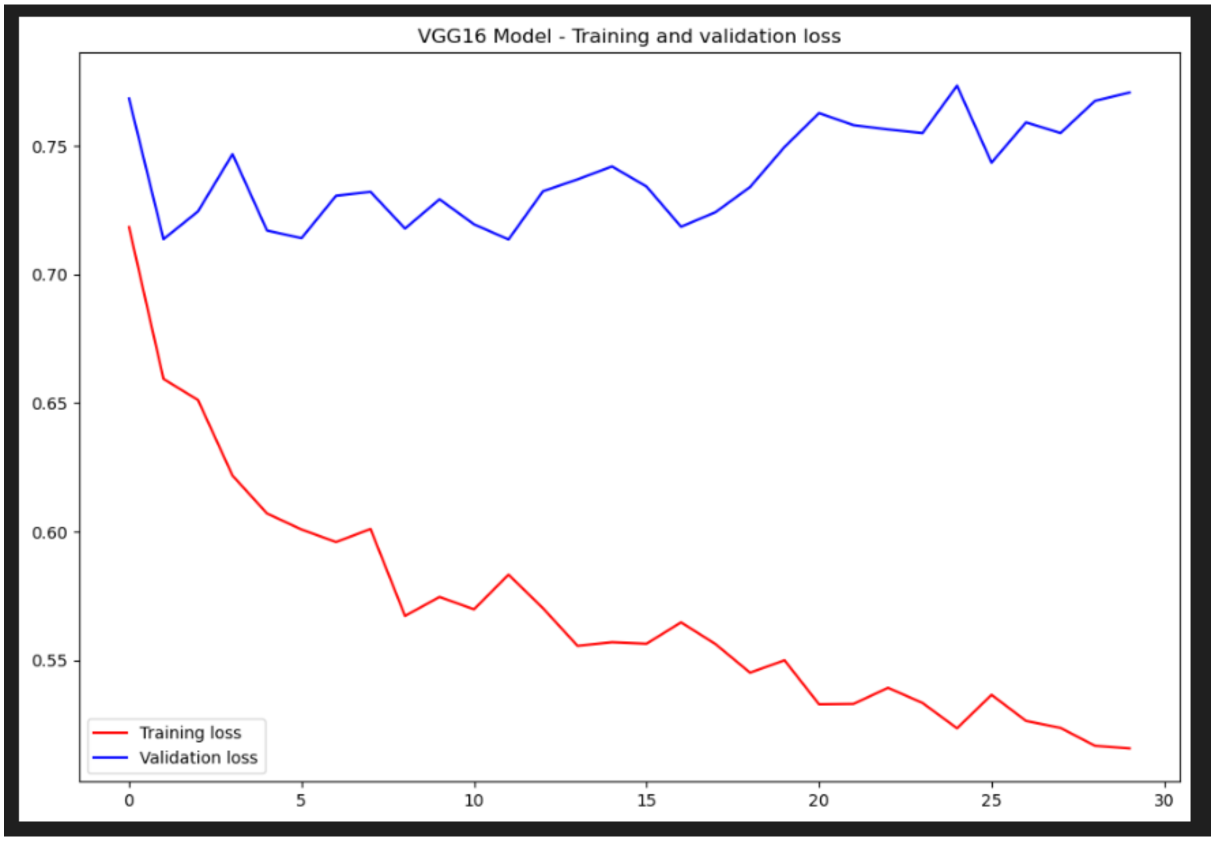
Training and Validation Plot code

Training and Validation Output




CNN Model - Training and validation loss
Training loss
Validation loss
0.74
0.72
0.70
0.68
0.66
0
5
10
15
20
25
30


VGG16 Model - Training and validation accuracy
Training accuracy
Validation accuracy
0.75
0.70
0.65
0.60
0.55
0
5
10
15
20
25
30



Proposal of the best model

- **Best Model**: VGG16 Model
    - **Reasoning**:
        - **Higher Accuracy**: Demonstrated better overall performance in terms of accuracy.
        - **Lower Loss**: Indicated more effective learning and generalization.
        - **Better Metrics**: Superior precision, recall, and F1-score in classification reports.
        - **Fewer Misclassifications**: Reduced number of false positives and false negatives.

Conclusion:

In conclusion, after thorough evaluation of both the Convolutional Neural Network (CNN) and the VGG16 transfer learning model, we determined that the VGG16 model is the best choice for the real and fake face detection task. The VGG16 model, with its pre-trained weights from the ImageNet dataset, demonstrated higher accuracy and better performance metrics, including precision and recall, compared to the custom CNN model. Although the VGG16 model required more training time due to its complex architecture, this was partially mitigated by freezing the initial layers, and the resultant performance gains justified the additional computational effort. Thus, the VGG16 model's enhanced ability to generalize and accurately classify real and fake faces makes it the optimal model for this project.