CS6005

Deep Learning

Mini-Project Assignment - 2

Image Classification of Monkey Species using Transfer Learning

Steven F. Gilbert

2018103071

'P' Batch

11.05.21

Introduction

This Mini project aims to implement image classification on a suitable dataset using a method called Transfer Learning. Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

Problem Statement

The problem taken is here is to classify the different images of monkeys based on their species. To implement Transfer Learning here we implement a trained model, VGG16 which was proposed by researchers from University of Oxford. The aim here is to use the VGG16 model with its pre-trained weights as a feature extractor and to add only our own fully- connected layers and classifier pertaining to the dataset being used. Moreover, to clearly show the ability to transfer a small dataset of only less than 150 images per class is chosen. A pre-trained model will eliminate the need to train the model from scratch.

Dataset Details

This dataset contains 10 classes of monkey species, each corresponding to a species from Wikipedia's monkey cladogram. Images are 400x300 px or larger and .jpeg format (almost 1400 images), with less than 150 images per class. Images were downloaded with help of the googliser open source code. This dataset was intended as a test case for fine-grain classification tasks, perhaps best used in combination with transfer learning.[1]

Sample Images:

Mantled Howler

Patas Monkey

Bald Uakari

Japanese Macaque

Modules

The project code can be split into multiple modules, each with a specific function.

1) Reading the dataset

The dataset is set into two folders for training and validation. Here there are 10 classes present with very few images in each class, thus making it ideal for transfer learning. The size of images ranges from 400 x 300px or higher.

2) Preprocessing data

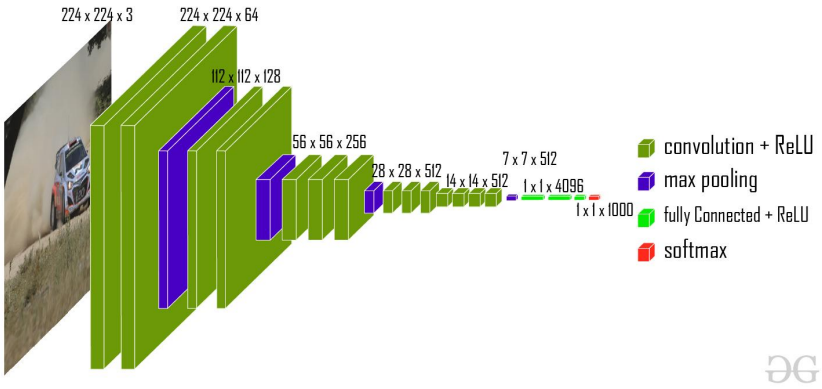
We then ImageDataGenerator from Tensorflow rescale the images at a value of 1/255. We then use the associated functions to split the train and validation sets. Since we are using the VGG16 model, the images must be resized to 254 x 254 px. Moreover we use VGG16's preprocess function available in Keras. Training batch size is 32 and validation is 10. The class mode is categorical.

3) Model Creation

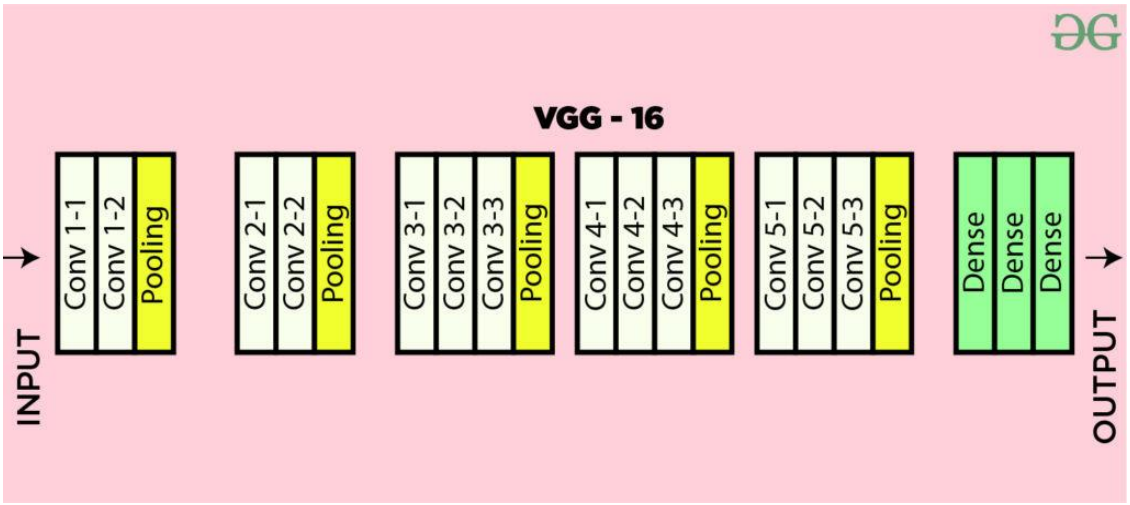
As previously explained, to implement transfer learning we employ the VGG16 model and use the pre-trained weights of the model. Using VGG16 as the feature extractor, we build upon it by adding a Global Average Pooling layer to minimize overfitting by reducing the parameters and a batch normalization layer to increase the training times and reduce epochs. After a Dense layer with 128 units

and ‘relu’ activation function is added after which a dropout layer at 0.5 is added to prevent overfitting. The output layer has 10 units as there are 10 classes and ‘softmax’ activation function.

VGG16



VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014. This model achieves 92.7% top-5 test accuracy on ImageNet dataset which contains 14 million images belonging to 1000 classes. The ImageNet dataset contains images of fixed size of 224*224 and have RGB channels. So, we have a tensor of (224, 224, 3) as our input. This model processes the input image and outputs a vector of 1000 values.



Libraries

The main reason why python is very popular in the field of deep learning is that it boasts a multitude of libraries to perform several important and painstaking modules and functions.

Some of the libraries used in this project are as follows:

1) Tensorflow - TensorFlow is an end-to-end open source platform for machine learning .

2) Keras - It offers consistent & simple APIs, it minimizes the number of user actions required for common use cases .

3) Numpy - Package which provides data computational functions .

4) Matplotlib - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python .

Model Summary

To solve the image classification problem we make use of VGG16 as the pretrained feature extractor and build upon it by adding fully connected layers and a classification layer for our dataset. A batch size of 32 is used for training the dataset. The model is trained for 35 epochs with early stop monitoring validation loss.

VGG16 architecture components :

1) **Convolution Layer :**
    a) The convolution layer allows the network to learn filters which maximally respond to a local region of the input.
    b) A pixel increases in correlation to nearby pixels than distant pixels.
    c) It uses a filter or kernel to produce a feature map.

2) **Max Pooling Layer :**
    a) Pooling layers reduce the dimensions of the feature map,thus no.of parameters to learn are also reduced.
    b) This layer summarizes the features of a region in a feature map from a convoluted layer
    c) A Max Pooling layer extracts the maximum value of the area it convolves.

User added components :

3) Global Average Pooling Layer.

a) Generates one feature map for each corresponding category.

b) No parameter required so overfitting is avoided.

4) Dropout Layer :

a) Dropout is a technique used to prevent the model from overfitting.

b) It works by randomly setting the outgoing edges of hidden units to 0 at each update of the phase.

5) Dense Layer :

a) The dense layer performs functions on the input and produces an output vector.

b) It represents the layer which performs matrix vector multiplication.

<u>Model Architecture</u> :

Parameter	Value
Epochs	35 (early stopped at 24)
Optimizer	Adam
Learning Rate	0.001
Loss function	Categorical Cross Entropy

Dense Layer:

a) Units = 128

b) Dropout = 0.5

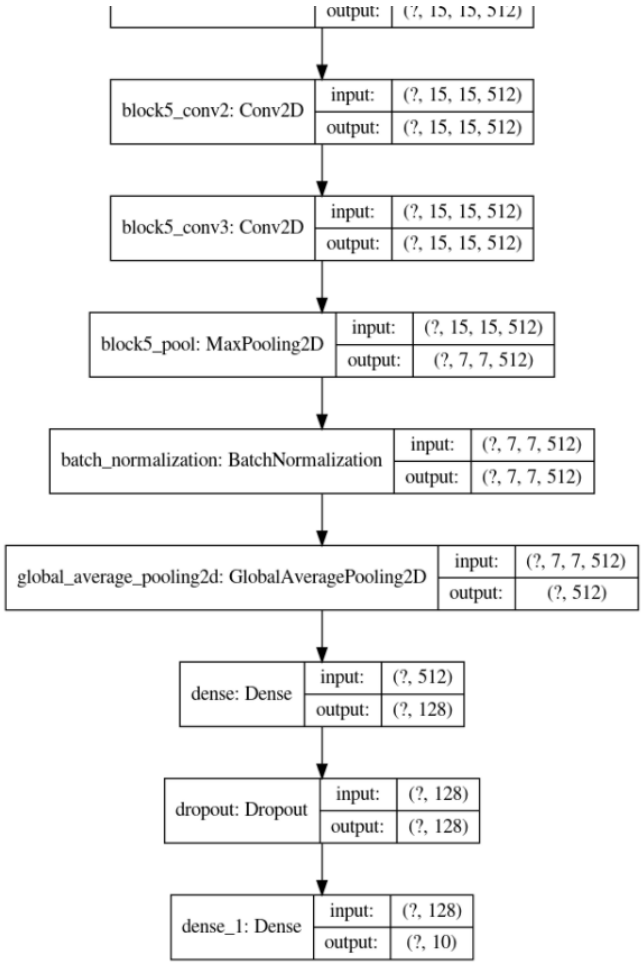
c) Activation function = relu

Output Layer:

a) Nodes = 10

b) Activation function = softmax

Model Summary



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 254, 254, 3)]	0
block1_conv1 (Conv2D)	(None, 254, 254, 64)	1792
block1_conv2 (Conv2D)	(None, 254, 254, 64)	36928
block1_pool (MaxPooling2D)	(None, 127, 127, 64)	0
block2_conv1 (Conv2D)	(None, 127, 127, 128)	73856
block2_conv2 (Conv2D)	(None, 127, 127, 128)	147584
block2_pool (MaxPooling2D)	(None, 63, 63, 128)	0
block3_conv1 (Conv2D)	(None, 63, 63, 256)	295168
block3_conv2 (Conv2D)	(None, 63, 63, 256)	590080
block3_conv3 (Conv2D)	(None, 63, 63, 256)	590080
block3_pool (MaxPooling2D)	(None, 31, 31, 256)	0
block4_conv1 (Conv2D)	(None, 31, 31, 512)	1180160
block4_conv2 (Conv2D)	(None, 31, 31, 512)	2359808
block4_conv3 (Conv2D)	(None, 31, 31, 512)	2359808
block4_pool (MaxPooling2D)	(None, 15, 15, 512)	0
block5_conv1 (Conv2D)	(None, 15, 15, 512)	2359808
block5_conv2 (Conv2D)	(None, 15, 15, 512)	2359808
block5_conv3 (Conv2D)	(None, 15, 15, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
batch_normalization (BatchNo	(None, 7, 7, 512)	2048
global_average_pooling2d (Gl	(None, 512)	0
dense (Dense)	(None, 128)	65664
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 14,783,690
Trainable params: 67,978
Non-trainable params: 14,715,712

Coding Snapshots

1) Importing Packages

```
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import tensorflow.keras as keras
from keras.applications.vgg16 import preprocess_input
from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
from keras.applications.vgg16 import VGG16
from keras.models import Model
from tensorflow.keras.layers import Dense,Input,GlobalAveragePooling2D,Dropout,BatchNormalization
```

2) Using Tensorflow ImageDataGenerator for rescaling

```
train_datagen=ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,preprocessing_function=preprocess_input)
val_datagen=ImageDataGenerator(rescale=1/255,preprocessing_function=preprocess_input)
```

3) Using Datagen to split train and test data splits and encoding categorical

```
train=train_datagen.flow_from_directory(
        directory='../input/10-monkey-species/training/training',
        class_mode='categorical',
        target_size=(254,254),
        batch_size=32
)
```

```
Found 1098 images belonging to 10 classes.
```

```
val=val_datagen.flow_from_directory(
        directory='../input/10-monkey-species/validation/validation',
        class_mode='categorical',
        target_size=(254,254),
        batch_size=10,shuffle=False
)
```

```
Found 272 images belonging to 10 classes.
```

4) Adding Pre-trained Model VGG16 using ‘imagenet’ weights and setting trainable to False.

```
pretrain = VGG16(include_top=False, weights='imagenet', input_shape=(254,254,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

```
for layer in pretrain.layers:
    layer.trainable = False
```

5) Adding upon pre-trained model

```
features = pretrain.output
bn = BatchNormalization()(features)
gp = GlobalAveragePooling2D()(bn)
d1 = Dense(128, activation = 'relu')(gp)
drop1 = Dropout(0.5)(d1)
output = Dense(10, activation='softmax')(drop1)
```

6) Model compile

```
from tensorflow.keras.optimizers import Adam
model.compile(
  loss='categorical_crossentropy',
  optimizer=Adam(learning_rate=0.001),
  metrics=['accuracy']
)
```

7) Early Stopping

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss',verbose=1,patience = 5,
                           mode='min', restore_best_weights=True)
```

8) Model training

```
history = model.fit_generator(generator=train,validation_data=val,epochs=40,shuffle=True,callbacks = [early_stop])
```

9) Plotting curves

```
import matplotlib.pyplot as plt
def plotLearningCurve(history,epochs):
  epochRange = range(1,epochs+1)
  plt.plot(epochRange,history.history['accuracy'])
  plt.plot(epochRange,history.history['val_accuracy'])
  plt.title('Model Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend(['Train','Validation'],loc='upper left')
  plt.show()

  plt.plot(epochRange,history.history['loss'])
  plt.plot(epochRange,history.history['val_loss'])
  plt.title('Model Loss')
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend(['Train','Validation'],loc='upper left')
  plt.show()
```

10) Model Evaluation

```
score = model.evaluate_generator(val,len(val))
```

Results

1) Splitting Train and Test data

```
Found 1098 images belonging to 10 classes.

Found 272 images belonging to 10 classes.
```

2) Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 254, 254, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 254, 254, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 254, 254, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 127, 127, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 127, 127, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 127, 127, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 63, 63, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 63, 63, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 63, 63, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 63, 63, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 31, 31, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 31, 31, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 31, 31, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 31, 31, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 15, 15, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 15, 15, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
batch_normalization (BatchNo (None, 7, 7, 512)         2048
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 14,783,690
Trainable params: 67,978
Non-trainable params: 14,715,712
_________________________________________________________________
```

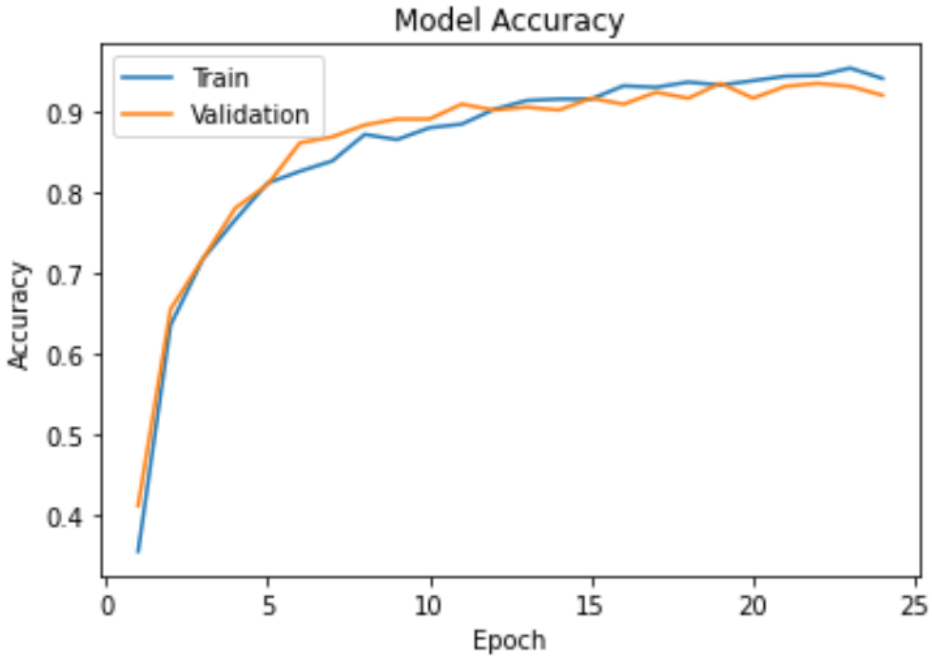
3) Training Model (early stopped at 24)

```
Epoch 1/40
35/35 [==============================] - 58s 2s/step - loss: 1.9147 - accuracy: 0.3552 - val_loss: 1.9516 - val_accuracy: 0.4118
Epoch 2/40
35/35 [==============================] - 44s 1s/step - loss: 1.2635 - accuracy: 0.6348 - val_loss: 1.6210 - val_accuracy: 0.6544
Epoch 3/40
35/35 [==============================] - 44s 1s/step - loss: 0.9430 - accuracy: 0.7168 - val_loss: 1.3233 - val_accuracy: 0.7169
Epoch 4/40
35/35 [==============================] - 44s 1s/step - loss: 0.7939 - accuracy: 0.7650 - val_loss: 1.0705 - val_accuracy: 0.7794
Epoch 5/40
35/35 [==============================] - 44s 1s/step - loss: 0.6451 - accuracy: 0.8106 - val_loss: 0.8645 - val_accuracy: 0.8088
Epoch 6/40
35/35 [==============================] - 44s 1s/step - loss: 0.5751 - accuracy: 0.8251 - val_loss: 0.7038 - val_accuracy: 0.8603
Epoch 7/40
35/35 [==============================] - 45s 1s/step - loss: 0.5069 - accuracy: 0.8379 - val_loss: 0.5890 - val_accuracy: 0.8676
Epoch 8/40
35/35 [==============================] - 45s 1s/step - loss: 0.4590 - accuracy: 0.8707 - val_loss: 0.4999 - val_accuracy: 0.8824
Epoch 9/40
35/35 [==============================] - 44s 1s/step - loss: 0.4179 - accuracy: 0.8643 - val_loss: 0.4454 - val_accuracy: 0.8897
Epoch 10/40
35/35 [==============================] - 44s 1s/step - loss: 0.4030 - accuracy: 0.8789 - val_loss: 0.4025 - val_accuracy: 0.8897
Epoch 11/40
35/35 [==============================] - 44s 1s/step - loss: 0.3656 - accuracy: 0.8834 - val_loss: 0.3597 - val_accuracy: 0.9081
Epoch 12/40
35/35 [==============================] - 44s 1s/step - loss: 0.3067 - accuracy: 0.9016 - val_loss: 0.3399 - val_accuracy: 0.9007
Epoch 13/40
35/35 [==============================] - 43s 1s/step - loss: 0.3066 - accuracy: 0.9126 - val_loss: 0.3152 - val_accuracy: 0.9044
Epoch 14/40
35/35 [==============================] - 45s 1s/step - loss: 0.2790 - accuracy: 0.9144 - val_loss: 0.3056 - val_accuracy: 0.9007
Epoch 15/40
35/35 [==============================] - 44s 1s/step - loss: 0.2741 - accuracy: 0.9144 - val_loss: 0.2948 - val_accuracy: 0.9154
Epoch 16/40
35/35 [==============================] - 44s 1s/step - loss: 0.2388 - accuracy: 0.9308 - val_loss: 0.2851 - val_accuracy: 0.9081
Epoch 17/40
35/35 [==============================] - 44s 1s/step - loss: 0.2292 - accuracy: 0.9290 - val_loss: 0.2774 - val_accuracy: 0.9228
Epoch 18/40
35/35 [==============================] - 43s 1s/step - loss: 0.2207 - accuracy: 0.9353 - val_loss: 0.2655 - val_accuracy: 0.9154
Epoch 19/40
35/35 [==============================] - 42s 1s/step - loss: 0.2202 - accuracy: 0.9317 - val_loss: 0.2535 - val_accuracy: 0.9338
Epoch 20/40
35/35 [==============================] - 43s 1s/step - loss: 0.2095 - accuracy: 0.9372 - val_loss: 0.2620 - val_accuracy: 0.9154
Epoch 21/40
35/35 [==============================] - 45s 1s/step - loss: 0.1986 - accuracy: 0.9426 - val_loss: 0.2644 - val_accuracy: 0.9301
Epoch 22/40
35/35 [==============================] - 44s 1s/step - loss: 0.1932 - accuracy: 0.9435 - val_loss: 0.2561 - val_accuracy: 0.9338
Epoch 23/40
35/35 [==============================] - 44s 1s/step - loss: 0.1676 - accuracy: 0.9526 - val_loss: 0.2620 - val_accuracy: 0.9301
Epoch 24/40
35/35 [==============================] - ETA: 0s - loss: 0.1952 - accuracy: 0.9399Restoring model weights from the end of the best epoch.
35/35 [==============================] - 44s 1s/step - loss: 0.1952 - accuracy: 0.9399 - val_loss: 0.2579 - val_accuracy: 0.9191
Epoch 00024: early stopping
```

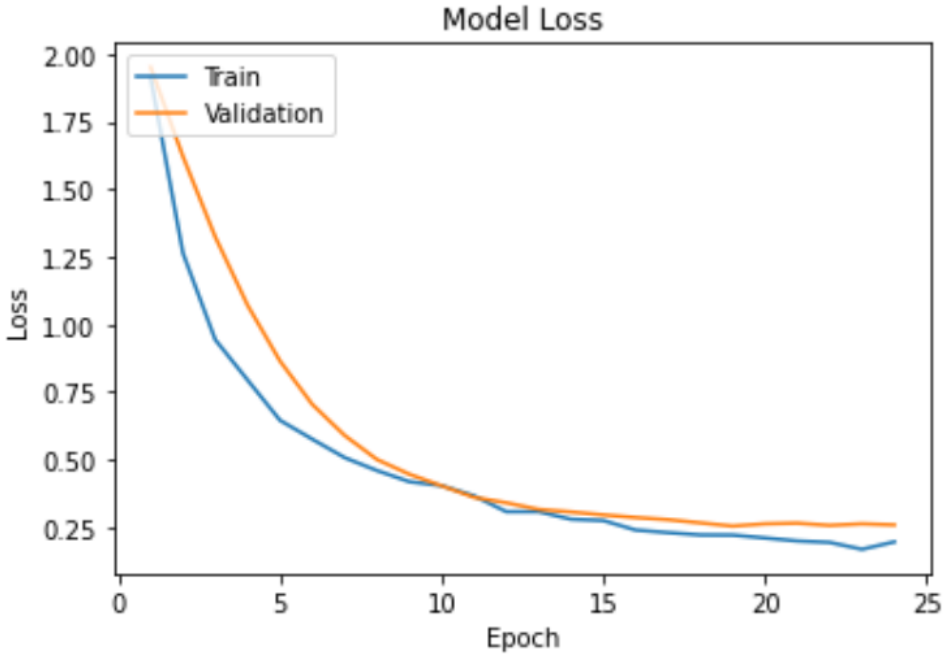
4) Final Validation Accuracy

```
Accuracy: 93.38%
Loss:  0.25350508093833923
```

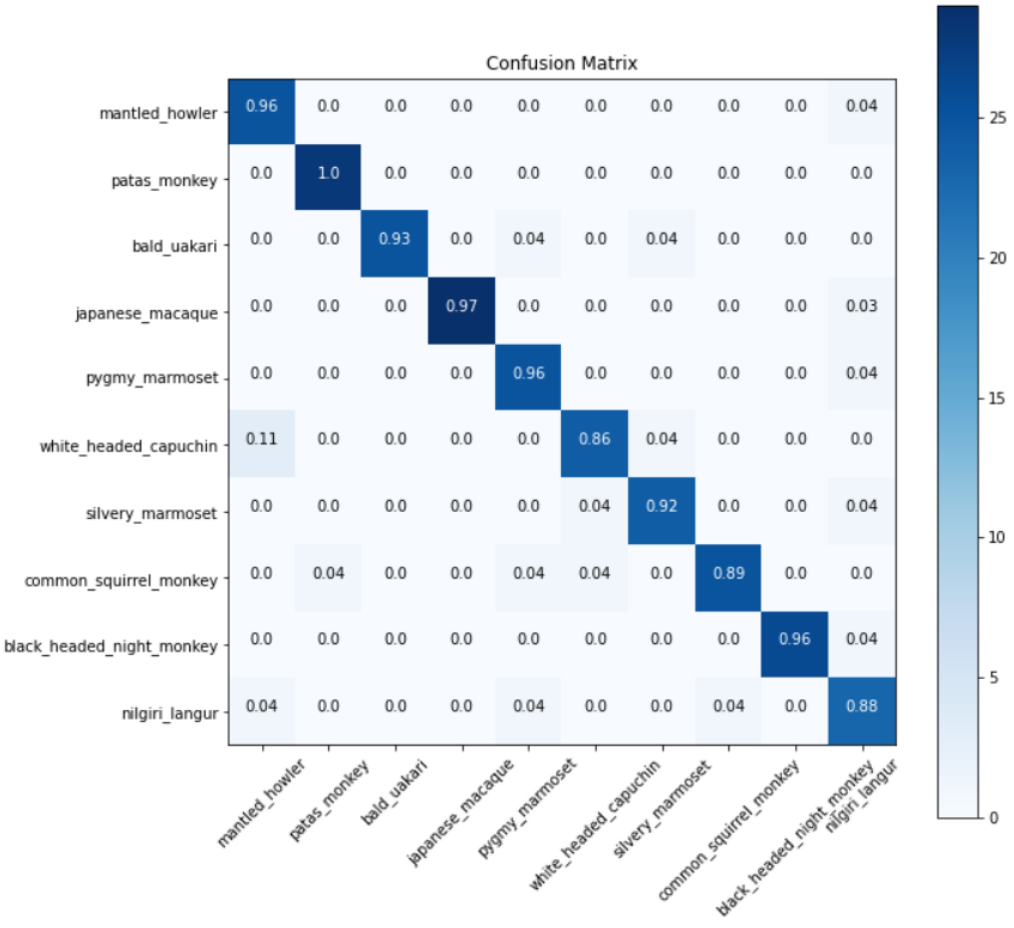
5) Learning Curve



6) Loss curve



7) Confusion Matrix



8) Classification Report

```
Classification Report
                          precision    recall  f1-score   support

          mantled_howler       0.86      0.96      0.91        26
            patas_monkey       0.97      1.00      0.98        28
             bald_uakari       1.00      0.93      0.96        27
         japanese_macaque       1.00      0.97      0.98        30
           pygmy_marmoset       0.89      0.96      0.93        26
     white_headed_capuchin       0.92      0.86      0.89        28
         silvery_marmoset       0.92      0.92      0.92        26
   common_squirrel_monkey       0.96      0.89      0.93        28
black_headed_night_monkey       1.00      0.96      0.98        27
          nilgiri_langur       0.82      0.88      0.85        26

                accuracy                           0.93       272
               macro avg       0.93      0.93      0.93       272
            weighted avg       0.94      0.93      0.93       272
```

Conclusion

As seen from the results, transfer learning has been achieved using VGG16 with its pre-trained weights and after early stopping for validation loss, the model was trained for 24 epochs. Accuracy of **93.38%** and Validation Loss of **0.2535**. Different aspects of transfer learning and the VGG16 model have been explained and its features have been understood.

References

[1] https://www.kaggle.com/slothkong/10-monkey-species