```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


from keras.applications.densenet import DenseNet121
from keras.applications.xception import Xception
from keras.applications.densenet import preprocess_input
from keras.optimizers import Adam, SGD, Adamax
from keras.models import Model, load_model
from keras.layers import *
from sklearn.model_selection import train_test_split
from keras.callbacks import *

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, Conv1D, MaxPool1D
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras import regularizers
from keras.src import callbacks
```

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
image_size = (224, 224)
batch_size = 32
```

```
train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    "/content/drive/MyDrive/BM1000",
    validation_split=0.25,
    subset="both",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 5850 files belonging to 7 classes.
Using 4388 files for training.
Using 1462 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
num_classes = len(class_names)
print(num_classes)
```

```
['BAS', 'EOS', 'HAC', 'LYT', 'MON', 'NGB', 'NGS']
7
```

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(224,
                                  224,
                                  3)),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2)
  ]
)
```

```
train_ds = train_ds.map(
  lambda x, y: (data_augmentation(x, training=True), y))
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
import time
### Define a class for custom callback
class MyCallback(keras.callbacks.Callback):
    def __init__(self, model, base_model, patience, stop_patience, threshold, factor, batches, initial_epoch, epochs, ask_epoch):
        super(MyCallback, self).__init__()
        self.model = model
        self.base_model = base_model
        self.patience = patience # specifies how many epochs without improvement before learning rate is adjusted
        self.stop_patience = stop_patience # specifies how many times to adjust lr without improvement to stop training
        self.threshold = threshold # specifies training accuracy threshold when lr will be adjusted based on validation loss
        self.factor = factor # factor by which to reduce the learning rate
        self.batches = batches # number of training batch to runn per epoch
        self.initial_epoch = initial_epoch
        self.epochs = epochs
        self.ask_epoch = ask_epoch
        self.ask_epoch_initial = ask_epoch # save this value to restore if restarting training
        # callback variables
        self.count = 0 # how many times lr has been reduced without improvement
        self.stop_count = 0
        self.best_epoch = 1   # epoch with the lowest loss
        self.initial_lr = float(tf.keras.backend.get_value(model.optimizer.lr)) # get the initial learning rate and save it
        self.highest_tracc = 0.0 # set highest training accuracy to 0 initially
        self.lowest_vloss = np.inf # set lowest validation loss to infinity initially
        self.best_weights = self.model.get_weights() # set best weights to model's initial weights
        self.initial_weights = self.model.get_weights()   # save initial weights if they have to get restored

    # Define a function that will run when train begins
    def on_train_begin(self, logs= None):
      msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_loss', 'V_acc', 'LR', '
      print(msg)
      self.start_time = time.time()

    def on_train_end(self, logs= None):
        stop_time = time.time()
        tr_duration = stop_time - self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} minutes, {seconds:4.2f} seconds)'
        print(msg)
```

```
        print(msg)
        self.model.set_weights(self.best_weights) # set the weights of the model to the best weights

    def on_train_batch_end(self, batch, logs= None):
        acc = logs.get('accuracy') * 100 # get batch accuracy
        loss = logs.get('loss')
        msg = '{0:20s}processing batch {1:} of {2:5s}-   accuracy=  {3:5.3f}   -   loss: {4:8.5f}'.format(' ', str(batch), str(self.batches), acc, los
        print(msg, '\r', end= '') # prints over on the same line to show running batch count

    def on_epoch_begin(self, epoch, logs= None):
        self.ep_start = time.time()

    # Define method runs on the end of each epoch
    def on_epoch_end(self, epoch, logs= None):
        ep_end = time.time()
        duration = ep_end - self.ep_start

        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get the current learning rate
        current_lr = lr
        acc = logs.get('accuracy')  # get training accuracy
        v_acc = logs.get('val_accuracy')  # get validation accuracy
        loss = logs.get('loss')  # get training loss for this epoch
        v_loss = logs.get('val_loss')  # get the validation loss for this epoch

        if acc < self.threshold: # if training accuracy is below threshold adjust lr based on training accuracy
            monitor = 'accuracy'
            if epoch == 0:
                pimprov = 0.0
            else:
                pimprov = (acc - self.highest_tracc ) * 100 / self.highest_tracc # define improvement of model progres

            if acc > self.highest_tracc: # training accuracy improved in the epoch
                self.highest_tracc = acc # set new highest training accuracy
                self.best_weights = self.model.get_weights() # training accuracy improved so save the weights
                self.count = 0 # set count to 0 since training accuracy improved
                self.stop_count = 0 # set stop counter to 0
                if v_loss < self.lowest_vloss:
                    self.lowest_vloss = v_loss
                self.best_epoch = epoch + 1  # set the value of best epoch for this epoch

            else:
                # training accuracy did not improve check if this has happened for patience number of epochs
                # if so adjust learning rate
                if self.count >= self.patience - 1: # lr should be adjusted
                    lr = lr * self.factor # adjust the learning by factor
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate in the optimizer
                    self.count = 0 # reset the count to 0
                    self.stop_count = self.stop_count + 1 # count the number of consecutive lr adjustments
                    self.count = 0 # reset counter
                    if v_loss < self.lowest_vloss:
                        self.lowest_vloss = v_loss
                else:
                    self.count = self.count + 1 # increment patience counter

        else: # training accuracy is above threshold so adjust learning rate based on validation loss
            monitor = 'val_loss'
            if epoch == 0:
                pimprov = 0.0
            else:
                pimprov = (self.lowest_vloss - v_loss ) * 100 / self.lowest_vloss
            if v_loss < self.lowest_vloss: # check if the validation loss improved
                self.lowest_vloss = v_loss # replace lowest validation loss with new validation loss
                self.best_weights = self.model.get_weights() # validation loss improved so save the weights
                self.count = 0 # reset count since validation loss improved
                self.stop_count = 0
                self.best_epoch = epoch + 1 # set the value of the best epoch to this epoch
            else: # validation loss did not improve
                if self.count >= self.patience - 1: # need to adjust lr
                    lr = lr * self.factor # adjust the learning rate
                    self.stop_count = self.stop_count + 1 # increment stop counter because lr was adjusted
                    self.count = 0 # reset counter
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate in the optimizer
                else:
                    self.count = self.count + 1 # increment the patience counter
                if acc > self.highest_tracc:
                    self.highest_tracc = acc

        msg = f'{str(epoch + 1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc * 100:^9.3f}{v_loss:^9.5f}{v_acc * 100:^9.3f}{current_lr:^9.5f}{lr:^9.5f}-
        print(msg)

        if self.stop_count > self.stop_patience - 1: # check if learning rate has been adjusted stop_count times with no improvement
            msg = f' training has been halted at epoch {epoch + 1} after {self.stop_patience} adjustments of learning rate with no improvement'
            print(msg)
            self.model.stop_training = True # stop training

        else:
            if self.ask_epoch != None:
                if epoch + 1 >= self.ask_epoch:
                    msg = 'enter H to halt training or an integer for number of epochs to run then ask again'
                    print(msg)
                    ans = input('')
                    if ans == 'H' or ans == 'h':
                        msg = f'training has been halted at epoch {epoch + 1} due to user input'
                        print(msg)
                        self.model.stop_training = True # stop training
                    else:
                        try:
                            ans = int(ans)
                            self.ask_epoch += ans
                            msg = f' training will continue until epoch ' + str(self.ask_epoch)
                            print(msg)
                            msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_lo
                            print(msg)
                        except:
                            print('Invalid')
```

ResNet50

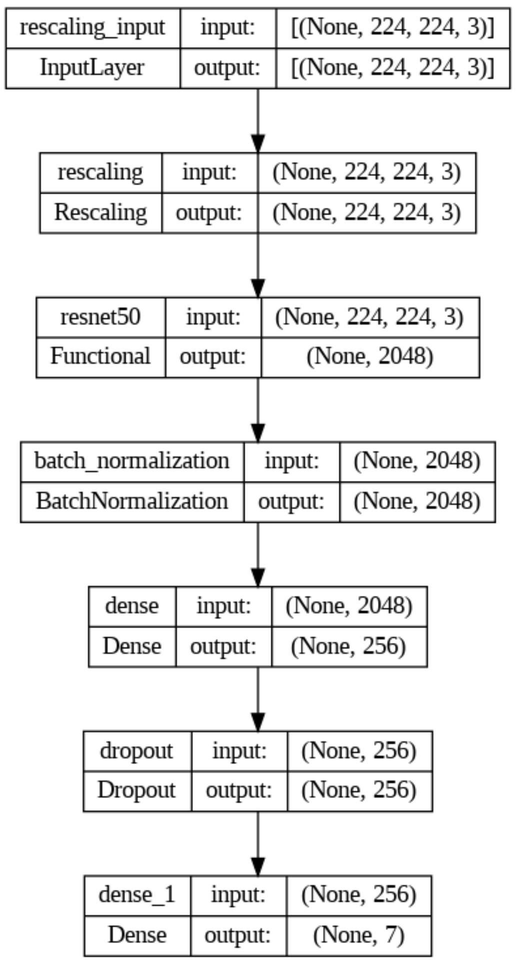
```
base_model = tf.keras.applications.ResNet50(include_top= False, weights= "imagenet", input_shape= (224,224,3), pooling= 'max')
```

```
model = Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    base_model,
    BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
    Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer= regularizers.l1(0.006),
                bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
    Dropout(rate= 0.45, seed= 123),
    Dense(7, activation= 'softmax')
```

```
])

model.compile(Adamax(learning_rate= 0.001, weight_decay=0.02), loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), metrics= ['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 resnet50 (Functional)       (None, 2048)              23587712

 batch_normalization (Batch  (None, 2048)              8192
 Normalization)

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 24122247 (92.02 MB)
Trainable params: 24065031 (91.80 MB)
Non-trainable params: 57216 (223.50 KB)
_________________________________________________________________
```

```
keras.utils.plot_model(model, show_shapes=True)
```

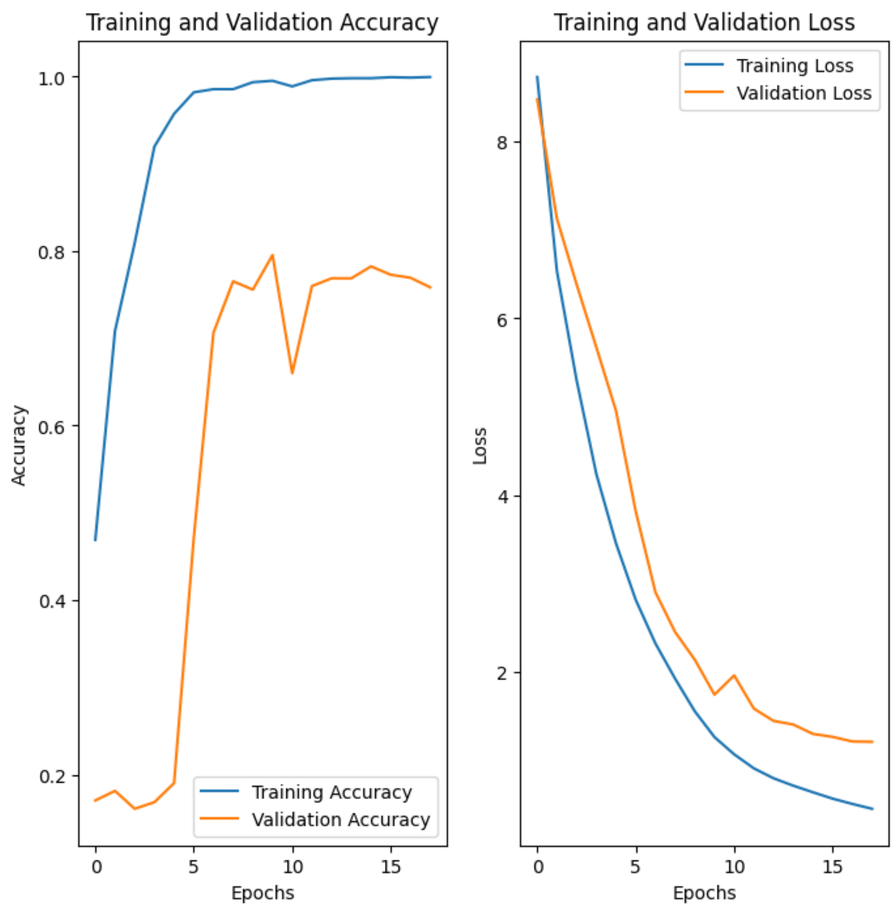



```
batch_size = 40
epochs = 40
patience = 1        # number of epochs to wait to adjust lr if monitored value does not improve
stop_patience = 3   # number of epochs to wait before stopping training if monitored value does not improve
threshold = 0.9     # if train accuracy is < threshhold adjust monitor accuracy, else monitor validation loss
factor = 0.5        # factor to reduce lr by
freeze = False      # if true free weights of  the base model
ask_epoch = 5       # number of epochs to run before asking if you want to halt training
#batches = int(np.ceil(len(train_ds.labels) / batch_size))
batches = int(np.ceil(4388 / batch_size))

callbacks = [MyCallback(model= model, base_model= base_model, patience= patience,
            stop_patience= stop_patience, threshold= threshold, factor= factor,
            batches= batches, initial_epoch= 0, epochs= epochs, ask_epoch= ask_epoch )]



history = model.fit(x= train_ds, epochs= epochs, verbose= 0, callbacks= callbacks,
                    validation_data= val_ds, validation_steps= None, shuffle= False,
                    initial_epoch= 0)
```

```
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.1422s vs `on_train_batch_end` time: 0.1554s). Check your callbacks.
 1 /40     8.735   46.901   8.48206  17.031   0.00100  0.00100  accuracy     0.00    321.55
 2 /40     6.536   70.852   7.12956  18.126   0.00100  0.00100  accuracy    51.07     52.00
 3 /40     5.299   80.925   6.38555  16.074   0.00100  0.00100  accuracy    14.22     51.20
 4 /40     4.238   91.978   5.67335  16.826   0.00100  0.00100  val_loss    11.15     51.28
 5 /40     3.452   95.738   4.95378  19.015   0.00100  0.00100  val_loss    12.68     51.16
enter H to halt training or an integer for number of epochs to run then ask again
2
 training will continue until epoch 7
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /40     2.813   98.200   3.81602  47.127   0.00100  0.00100  val_loss    22.97     53.78
 7 /40     2.318   98.564   2.89935  70.657   0.00100  0.00100  val_loss    24.02     50.88
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 12
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 8 /40     1.919   98.564   2.44764  76.539   0.00100  0.00100  val_loss    15.58     52.22
 9 /40     1.550   99.362   2.13390  75.581   0.00100  0.00100  val_loss    12.82     51.18
10 /40     1.257   99.521   1.73816  79.549   0.00100  0.00100  val_loss    18.55     51.96
11 /40     1.062   98.883   1.95570  66.005   0.00100  0.00050  val_loss   -12.52     52.05
12 /40     0.903   99.590   1.57993  75.992   0.00050  0.00050  val_loss     9.10     50.51
enter H to halt training or an integer for number of epochs to run then ask again
2
 training will continue until epoch 14
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
13 /40     0.791   99.772   1.44027  76.881   0.00050  0.00050  val_loss     8.84     52.13
14 /40     0.707   99.818   1.39952  76.881   0.00050  0.00050  val_loss     2.83     51.64
enter H to halt training or an integer for number of epochs to run then ask again
2
 training will continue until epoch 16
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
15 /40     0.632   99.818   1.29323  78.249   0.00050  0.00050  val_loss     7.59     52.08
```

```
16 /40     0.560   99.932   1.25949  77.291   0.00050  0.00050  val_loss     2.61    51.10
enter H to halt training or an integer for number of epochs to run then ask again
2
 training will continue until epoch 18
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
17 /40     0.500   99.886   1.20782  76.949   0.00050  0.00050  val_loss     4.10    52.28
18 /40     0.445   99.954   1.20356  75.855   0.00050  0.00050  val_loss     0.35    51.20
enter H to halt training or an integer for number of epochs to run then ask again
H
training has been halted at epoch 18 due to user input
training elapsed time was 0.0 hours, 30.0 minutes, 8.36 seconds)
```

```
#Create plots of the loss and accuracy on the training and validation sets:
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title('Training and Validation Loss')
plt.show()
```




```
y_test = np.concatenate([y for x, y in val_ds], axis=0)
y_pred = model.predict(val_ds)
y_pred_classes = np.argmax(y_pred, axis=1)
accuracy_score(y_test, y_pred_classes)
```

```
46/46 [==============================] - 5s 98ms/step
0.7585499316005472
```

```
print(classification_report(y_test, y_pred_classes, target_names=class_names))
```

```
              precision    recall  f1-score   support

         BAS       0.78      0.56      0.66       110
         EOS       0.82      0.94      0.87       255
         HAC       0.73      0.77      0.75        93
         LYT       0.94      0.83      0.88       265
         MON       0.73      0.81      0.76       255
         NGB       0.57      0.86      0.69       235
         NGS       0.92      0.43      0.59       249

    accuracy                           0.76      1462
   macro avg       0.78      0.74      0.74      1462
weighted avg       0.79      0.76      0.75      1462
```

VGG16

```
base_model = tf.keras.applications.VGG16(include_top= False, weights= "imagenet", input_shape= (224,224,3), pooling= 'max')
```

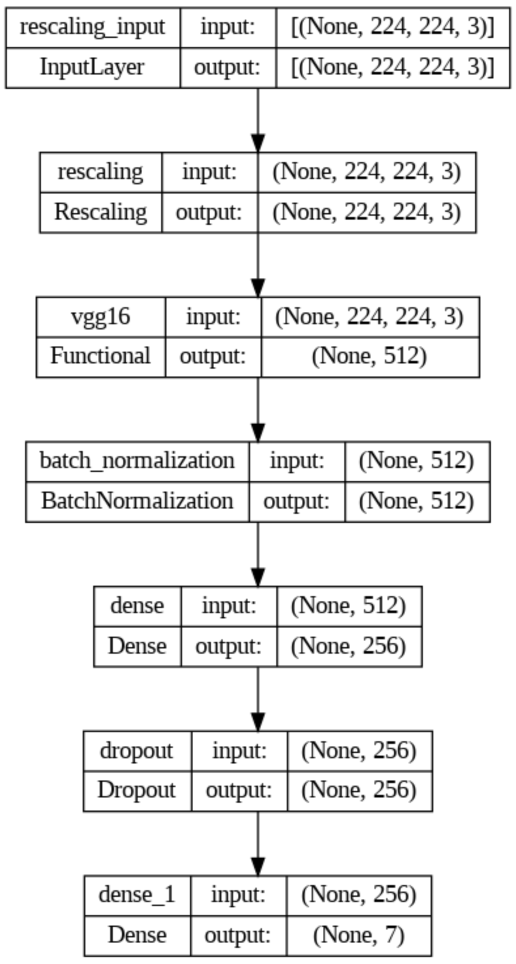
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
model = Sequential([
    layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    base_model,
    BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
    Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer= regularizers.l1(0.006),
            bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
    Dropout(rate= 0.45, seed= 123),
    Dense(7, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001, weight_decay=0.02), loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), metrics= ['accu
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 512)               14714688

 batch_normalization (Batch  (None, 512)               2048
 Normalization)

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 14849863 (56.65 MB)
Trainable params: 14848839 (56.64 MB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

```
keras.utils.plot_model(model, show_shapes=True)
```

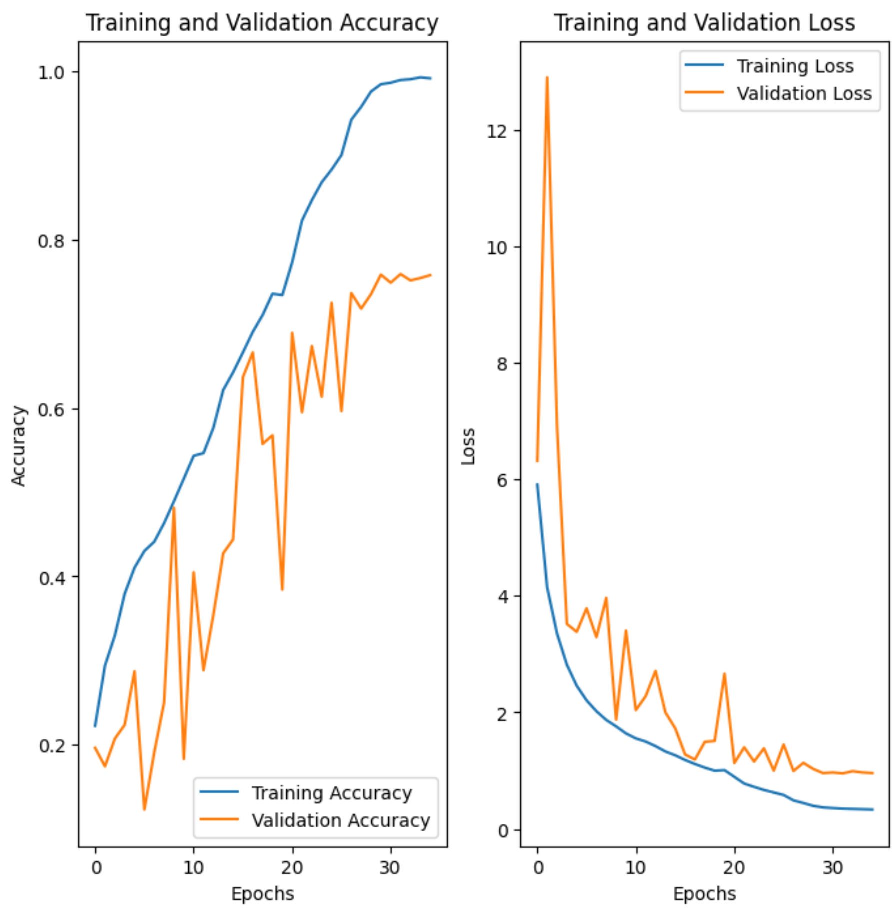



```
batch_size = 40
epochs = 40
patience = 1        # number of epochs to wait to adjust lr if monitored value does not improve
stop_patience = 3   # number of epochs to wait before stopping training if monitored value does not improve
threshold = 0.9     # if train accuracy is < threshhold adjust monitor accuracy, else monitor validation loss
factor = 0.5        # factor to reduce lr by
freeze = False      # if true free weights of  the base model
ask_epoch = 5       # number of epochs to run before asking if you want to halt training
#batches = int(np.ceil(len(train_ds.labels) / batch_size))
batches = int(np.ceil(4388 / batch_size))

callbacks = [MyCallback(model= model, base_model= base_model, patience= patience,
            stop_patience= stop_patience, threshold= threshold, factor= factor,
            batches= batches, initial_epoch= 0, epochs= epochs, ask_epoch= ask_epoch )]


history = model.fit(x= train_ds, epochs= epochs, verbose= 0, callbacks= callbacks,
                    validation_data= val_ds, validation_steps= None, shuffle= False,
                    initial_epoch= 0)
```

```
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.1402s vs `on_train_batch_end` time: 0.2424s). Check your callbacks.
 1 /40     5.907   22.265   6.31445  19.631   0.00100  0.00100  accuracy     0.00    153.23
 2 /40     4.143   29.421  12.89272  17.442   0.00100  0.00100  accuracy    32.14    64.76
 3 /40     3.353   32.976   6.90973  20.725   0.00100  0.00100  accuracy    12.08    68.61
 4 /40     2.818   37.944   3.51719  22.367   0.00100  0.00100  accuracy    15.07    64.66
 5 /40     2.456   41.044   3.38124  28.728   0.00100  0.00100  accuracy     8.17    65.34
enter H to halt training or an integer for number of epochs to run then ask again
7
 training will continue until epoch 12
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /40     2.206   43.026   3.78423  12.312   0.00100  0.00100  accuracy     4.83    65.31
 7 /40     2.019   44.120   3.29024  19.083   0.00100  0.00100  accuracy     2.54    64.36
 8 /40     1.869   46.331   3.96301  25.034   0.00100  0.00100  accuracy     5.01    64.52
 9 /40     1.759   48.929   1.87505  48.153   0.00100  0.00100  accuracy     5.61    68.61
10 /40     1.639   51.618   3.40146  18.331   0.00100  0.00100  accuracy     5.50    68.77
11 /40     1.554   54.330   2.03941  40.492   0.00100  0.00100  accuracy     5.25    64.75
12 /40     1.499   54.649   2.27889  28.865   0.00100  0.00100  accuracy     0.59    64.56
enter H to halt training or an integer for number of epochs to run then ask again
7
 training will continue until epoch 19
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
13 /40     1.421   57.657   2.71113  35.431   0.00100  0.00100  accuracy     5.50    65.58
14 /40     1.329   62.147   2.00028  42.750   0.00100  0.00100  accuracy     7.79    64.56
15 /40     1.263   64.243   1.72662  44.391   0.00100  0.00100  accuracy     3.37    64.58
16 /40     1.183   66.613   1.27748  63.680   0.00100  0.00100  accuracy     3.69    64.80
17 /40     1.114   69.052   1.19107  66.621   0.00100  0.00100  accuracy     3.66    64.50
18 /40     1.052   71.057   1.49422  55.746   0.00100  0.00100  accuracy     2.90    68.59
19 /40     1.000   73.587   1.51046  56.772   0.00100  0.00100  accuracy     3.56    64.95
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 24
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
20 /40     1.009   73.428   2.66314  38.440   0.00100  0.00050  accuracy    -0.22    69.75
21 /40     0.894   77.325   1.13187  68.947   0.00050  0.00050  accuracy     5.08    69.21
22 /40     0.778   82.293   1.40003  59.508   0.00050  0.00050  accuracy     6.42    64.80
23 /40     0.722   84.708   1.15629  67.373   0.00050  0.00050  accuracy     2.94    68.66
24 /40     0.669   86.828   1.38226  61.354   0.00050  0.00050  accuracy     2.50    68.80
enter H to halt training or an integer for number of epochs to run then ask again
```

```
 training will continue until epoch 34
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
25 /40     0.626   88.332   1.00064  72.503  0.00050  0.00050  accuracy     1.73     69.86
26 /40     0.583   90.064   1.44865  59.644  0.00050  0.00025  val_loss   -44.77     65.31
27 /40     0.490   94.257   0.99415  73.666  0.00025  0.00025  val_loss     0.65     68.88
28 /40     0.446   95.784   1.13367  71.819  0.00025  0.00013  val_loss   -14.03     64.97
29 /40     0.397   97.607   1.02996  73.529  0.00013  0.00006  val_loss    -3.60     64.59
30 /40     0.370   98.473   0.95787  75.855  0.00006  0.00006  val_loss     3.65     65.17
31 /40     0.358   98.655   0.96833  74.897  0.00006  0.00003  val_loss    -1.09     64.88
32 /40     0.348   98.974   0.95292  75.923  0.00003  0.00003  val_loss     0.52     68.81
33 /40     0.344   99.066   0.99069  75.171  0.00003  0.00002  val_loss    -3.96     65.21
34 /40     0.339   99.294   0.96877  75.445  0.00002  0.00001  val_loss    -1.66     65.79
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 39
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
35 /40     0.334   99.180   0.95819  75.787  0.00001  0.00000  val_loss    -0.55     65.81
 training has been halted at epoch 35 after 3 adjustments of learning rate with no improvement
training elapsed time was 0.0 hours, 46.0 minutes, 46.16 seconds)
```

```python
#Create plots of the loss and accuracy on the training and validation sets:
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title('Training and Validation Loss')
plt.show()
```




```python
y_test = np.concatenate([y for x, y in val_ds], axis=0)
y_pred = model.predict(val_ds)
y_pred_classes = np.argmax(y_pred, axis=1)
accuracy_score(y_test, y_pred_classes)
```

```
46/46 [==============================] - 6s 129ms/step
0.759233926128591
```

```python
print(classification_report(y_test, y_pred_classes, target_names=class_names))
```

```
              precision    recall  f1-score   support

         BAS       0.70      0.50      0.58       110
         EOS       0.85      0.91      0.88       255
         HAC       0.77      0.73      0.75        93
         LYT       0.90      0.81      0.85       265
         MON       0.72      0.75      0.73       255
         NGB       0.61      0.76      0.67       235
         NGS       0.77      0.68      0.72       249

    accuracy                           0.76      1462
   macro avg       0.76      0.74      0.74      1462
weighted avg       0.77      0.76      0.76      1462
```