Libraries

```
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
```

Train Test Split

```
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

```
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))
```

Configure the dataset for performance

Make sure to use buffered prefetching, so you can yield data from disk without having I/O become blocking. These are two important methods you should use when loading data:

Dataset.cache keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training your model. If your dataset is too large to fit into memory, you can also use this method to create a performant on-disk cache.

Dataset.prefetch overlaps data preprocessing and model execution while training.

```
AUTOTUNE = tf.data.AUTOTUNE

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

Custom Callback

```
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',
      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)
        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),
        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}{l
        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
                            print(msg)
                        except:
                            print('Invalid')
```

ConvNeXtTiny

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/convnext/convnext_tiny_notop.h5
111650432/111650432 [==============================] - 6s 0us/step
```

```
model = Sequential([
    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=

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 convnext_tiny (Functional)  (None, 768)               27820128

 batch_normalization (Batch  (None, 768)               3072
 Normalization)

 dense (Dense)               (None, 256)               196864

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 28021863 (106.89 MB)
Trainable params: 28020327 (106.89 MB)
Non-trainable params: 1536 (6.00 KB)
_________________________________________________________________
```

```
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
 1 /40     7.157   43.961   6.75650  17.442   0.00100  0.00100  accuracy     0.00   1891.00
 2 /40     4.994   69.690   5.07845  42.476   0.00100  0.00100  accuracy    58.53    110.89
 3 /40     3.644   80.356   3.67886  59.508   0.00100  0.00100  accuracy    15.30    110.99
 4 /40     2.627   87.694   2.70595  68.263   0.00100  0.00100  accuracy     9.13    110.82
 5 /40     1.846   93.596   2.12952  75.718   0.00100  0.00100  val_loss    21.30    111.24
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 10
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /40     1.272   97.493   1.70686  77.155   0.00100  0.00100  val_loss    19.85    110.87
 7 /40     0.870   99.658   1.36678  77.497   0.00100  0.00100  val_loss    19.92    110.76
 8 /40     0.618   99.932   1.19015  78.454   0.00100  0.00100  val_loss    12.92    120.46
 9 /40     0.462   99.954   1.10790  78.728   0.00100  0.00100  val_loss     6.91    111.17
10 /40     0.365   100.000  0.94855  80.506   0.00100  0.00100  val_loss    14.38    110.93
enter H to halt training or an integer for number of epochs to run then ask again
2
 training will continue until epoch 12
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
11 /40     0.299   100.000  0.91888  79.891   0.00100  0.00100  val_loss     3.13    111.38
12 /40     0.251   100.000  0.87932  81.327   0.00100  0.00100  val_loss     4.30    111.03
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     0.220   100.000  0.82648  81.327   0.00100  0.00100  val_loss     6.01    111.28
14 /40     0.216   99.681   0.92244  77.633   0.00100  0.00050  val_loss    -11.61   110.95
enter H to halt training or an integer for number of epochs to run then ask again
H
training has been halted at epoch 14 due to user input
training elapsed time was 1.0 hours,  2.0 minutes, 18.38 seconds)
```

```
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 [==============================] - 19s 259ms/step
0.8132694938440492
```

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

```
                  precision    recall  f1-score   support

             BAS       0.75      0.69      0.72       110
             EOS       0.95      0.89      0.92       255
             HAC       0.89      0.71      0.79        93
             LYT       0.92      0.91      0.92       265
             MON       0.75      0.85      0.80       255
             NGB       0.67      0.77      0.71       235
             NGS       0.80      0.72      0.76       249

        accuracy                           0.81      1462
       macro avg       0.82      0.79      0.80      1462
    weighted avg       0.82      0.81      0.81      1462
```

ConvNeXtSmall

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/convnext/convnext_small_notop.h5
198551472/198551472 [==============================] - 12s 0us/step
```

```
model = Sequential([
    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=

model.summary()
```

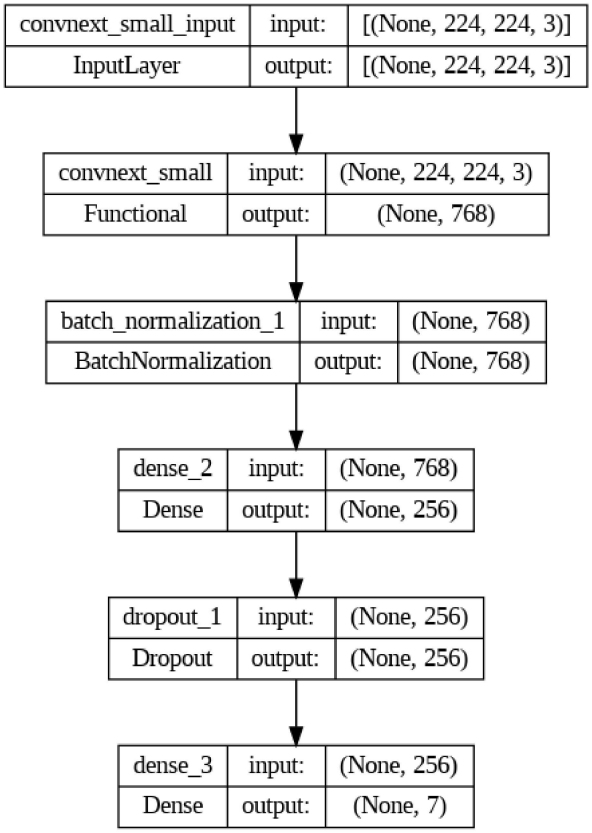
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 convnext_small (Functional  (None, 768)               49454688
 )

 batch_normalization_1 (Bat  (None, 768)               3072
 chNormalization)

 dense_2 (Dense)             (None, 256)               196864

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 7)                 1799

=================================================================
Total params: 49656423 (189.42 MB)
Trainable params: 49654887 (189.42 MB)
Non-trainable params: 1536 (6.00 KB)
_________________________________________________________________
```

```
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
 1 /40     7.622   41.431   6.53380  34.063   0.00100  0.00100  accuracy     0.00    291.37
 2 /40     5.280   73.108   5.13748  71.341   0.00100  0.00100  accuracy    76.46    188.71
 3 /40     3.958   84.731   3.80879  78.523   0.00100  0.00100  accuracy    15.90    188.18
 4 /40     2.963   91.158   2.92149  78.796   0.00100  0.00100  val_loss    23.30    188.01
 5 /40     2.194   95.784   2.34063  80.643   0.00100  0.00100  val_loss    19.88    190.23
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 10
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /40     1.594   98.883   2.00701  80.438   0.00100  0.00100  val_loss    14.25    188.27
 7 /40     1.176   99.590   1.64974  81.259   0.00100  0.00100  val_loss    17.80    190.17
 8 /40     0.869   99.863   1.48002  79.891   0.00100  0.00100  val_loss    10.29    188.34
 9 /40     0.660   99.909   1.13508  84.063   0.00100  0.00100  val_loss    23.31    188.11
10 /40     0.499   100.000  0.99732  84.679   0.00100  0.00100  val_loss    12.14    188.27
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 15
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
11 /40     0.454   98.291   0.99034  80.164   0.00100  0.00100  val_loss     0.70    188.51
12 /40     0.408   97.903   1.16150  77.360   0.00100  0.00050  val_loss    -17.28   188.46
13 /40     0.316   99.521   0.93059  82.900   0.00050  0.00050  val_loss     6.03    188.28
14 /40     0.267   100.000  0.83241  83.789   0.00050  0.00050  val_loss    10.55    188.40
15 /40     0.242   100.000  0.81382  83.653   0.00050  0.00050  val_loss     2.23    188.25
enter H to halt training or an integer for number of epochs to run then ask again
```

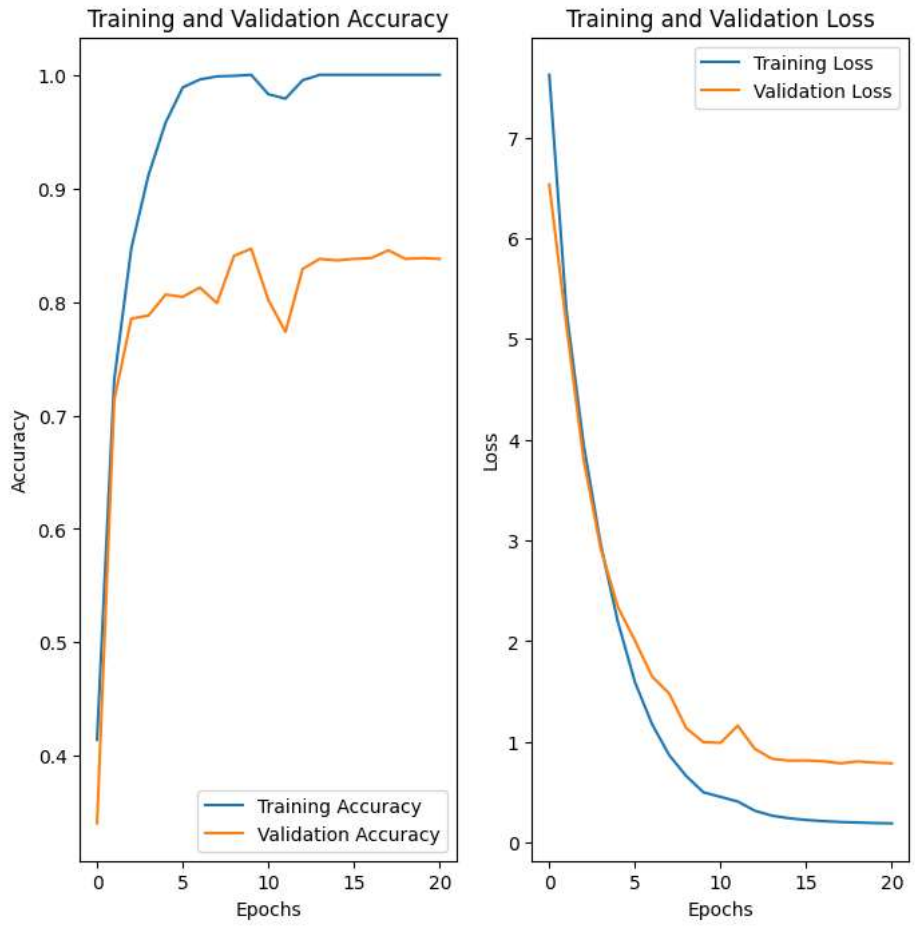
```
2
 training will continue until epoch 17
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
16 /40     0.225   100.000  0.81598  83.789   0.00050  0.00025  val_loss    -0.27    190.62
17 /40     0.213   100.000  0.80736  83.858   0.00025  0.00025  val_loss     0.79    188.96
enter H to halt training or an integer for number of epochs to run then ask again
1
 training will continue until epoch 18
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
18 /40     0.204   100.000  0.78687  84.542   0.00025  0.00025  val_loss     2.54    189.06
enter H to halt training or an integer for number of epochs to run then ask again
1
 training will continue until epoch 19
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
19 /40     0.199   100.000  0.80576  83.789   0.00025  0.00013  val_loss    -2.40    189.79
enter H to halt training or an integer for number of epochs to run then ask again
1
 training will continue until epoch 20
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
20 /40     0.194   100.000  0.79353  83.858   0.00013  0.00006  val_loss    -0.85    190.86
enter H to halt training or an integer for number of epochs to run then ask again
1
 training will continue until epoch 21
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
21 /40     0.191   100.000  0.78757  83.789   0.00006  0.00003  val_loss    -0.09    188.86
 training has been halted at epoch 21 after 3 adjustments of learning rate with no improvement
training elapsed time was 1.0 hours, 18.0 minutes, 3.50 seconds)
```

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




```
#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 [==============================] - 24s 419ms/step
0.8454172366621067
```

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

```
              precision    recall  f1-score   support

         BAS       0.87      0.69      0.77       110
         EOS       0.95      0.95      0.95       255
         HAC       0.86      0.76      0.81        93
         LYT       0.91      0.95      0.93       265
         MON       0.83      0.82      0.83       255
         NGB       0.76      0.78      0.77       235
         NGS       0.76      0.82      0.79       249

    accuracy                           0.85      1462
   macro avg       0.85      0.82      0.83      1462
weighted avg       0.85      0.85      0.84      1462
```