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.
```

```
train_ds
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int32, name=None))>
```

```
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 Callbacks

```
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}{li
        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')
```

DenseNet121

```
base_model=tf.keras.applications.DenseNet121(include_top= False, weights= "imagenet", input_shape= (224,224,3), pooling= 'max')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kerne
29084464/29084464 [==============================] - 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=

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 densenet121 (Functional)    (None, 1024)              7037504

 batch_normalization (Batch  (None, 1024)              4096
 Normalization)
```

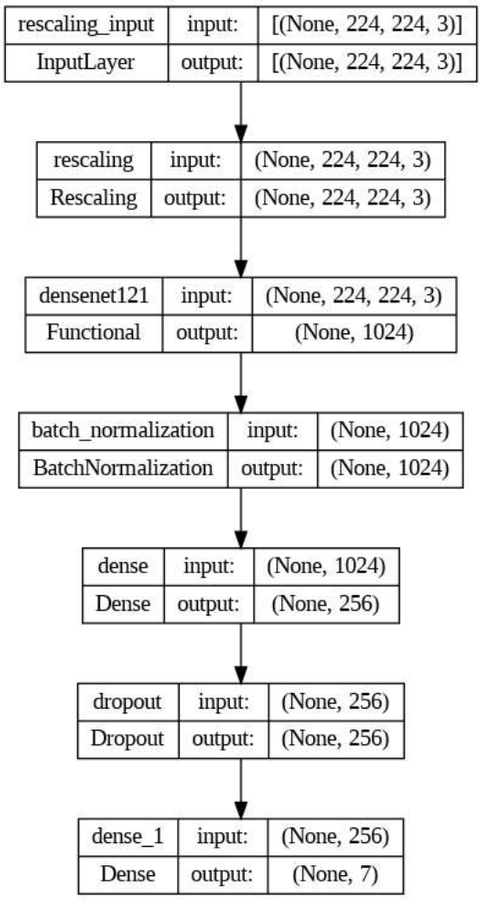
```
 dense (Dense)               (None, 256)               262400

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 7305799 (27.87 MB)
Trainable params: 7220103 (27.54 MB)
Non-trainable params: 85696 (334.75 KB)
_________________________________________________________________
```

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




```
train = np.concatenate([y for x, y in train_ds], axis=0)
len(train)
```

```
4388
```

```
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     6.633   39.198   6.60142  48.632   0.00100  0.00100  accuracy     0.00    560.09
 2 /40     4.334   61.987   3.70347  71.956   0.00100  0.00100  accuracy    58.14    225.13
 3 /40     3.350   70.214   2.84616  78.659   0.00100  0.00100  accuracy    13.27    224.53
 4 /40     2.684   74.499   2.44648  75.308   0.00100  0.00100  accuracy     6.10    219.65
 5 /40     2.255   76.185   2.04556  77.702   0.00100  0.00100  accuracy     2.26    225.51
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     1.915   78.077   1.82772  76.334   0.00100  0.00100  accuracy     2.48    224.72
 7 /40     1.664   78.829   1.64515  74.419   0.00100  0.00100  accuracy     0.96    224.78
 8 /40     1.468   78.965   1.36977  80.096   0.00100  0.00100  accuracy     0.17    228.07
 9 /40     1.303   80.059   1.36763  75.923   0.00100  0.00100  accuracy     1.39    228.67
10 /40     1.206   80.059   1.13806  80.096   0.00100  0.00050  accuracy     0.00    224.36
enter H to halt training or an integer for number of epochs to run then ask again
6
 training will continue until epoch 16
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
11 /40     1.055   82.566   0.99145  83.037   0.00050  0.00050  accuracy     3.13    222.71
12 /40     0.973   83.888   1.00610  81.874   0.00050  0.00050  accuracy     1.60    220.24
13 /40     0.924   84.868   0.93076  83.105   0.00050  0.00050  accuracy     1.17    220.42
```

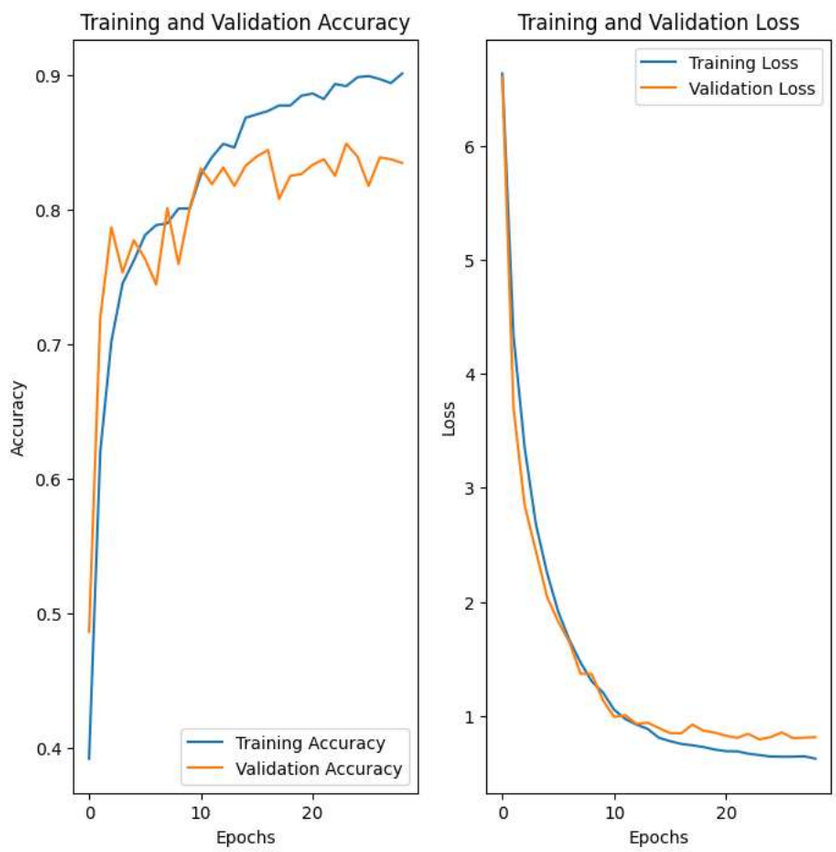
```
14 /40     0.886   84.594   0.94095  81.737   0.00050  0.00025  accuracy    -0.32    219.42
15 /40     0.808   86.805   0.89385  83.242   0.00025  0.00025  accuracy     2.28    221.06
16 /40     0.779   87.056   0.84956  83.926   0.00025  0.00025  accuracy     0.29    220.47
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 21
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
17 /40     0.755   87.306   0.84766  84.405   0.00025  0.00025  accuracy     0.29    219.54
18 /40     0.743   87.717   0.92366  80.780   0.00025  0.00025  accuracy     0.47    220.05
19 /40     0.727   87.717   0.87074  82.490   0.00025  0.00013  accuracy     0.00    221.27
20 /40     0.705   88.446   0.85363  82.627   0.00013  0.00013  accuracy     0.83    218.76
21 /40     0.691   88.605   0.82662  83.311   0.00013  0.00013  accuracy     0.18    219.18
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 26
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
22 /40     0.690   88.195   0.80776  83.721   0.00013  0.00006  accuracy    -0.46    224.47
23 /40     0.669   89.312   0.84311  82.490   0.00006  0.00006  accuracy     0.80    221.20
24 /40     0.657   89.152   0.79415  84.884   0.00006  0.00003  accuracy    -0.18    220.28
25 /40     0.644   89.813   0.81428  83.926   0.00003  0.00003  accuracy     0.56    218.35
26 /40     0.642   89.904   0.85606  81.737   0.00003  0.00003  accuracy     0.10    218.79
enter H to halt training or an integer for number of epochs to run then ask again
5
 training will continue until epoch 31
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
27 /40     0.643   89.676   0.80592  83.858   0.00003  0.00002  accuracy    -0.25    220.40
28 /40     0.646   89.380   0.80939  83.721   0.00002  0.00001  accuracy    -0.58    219.49
29 /40     0.626   90.087   0.81368  83.447   0.00001  0.00000  val_loss    -2.46    218.14
 training has been halted at epoch 29 after 3 adjustments of learning rate with no improvement
training elapsed time was 2.0 hours,  9.0 minutes, 57.85 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 [==============================] - 7s 103ms/step
0.8173734610123119
```

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

```
                  precision    recall  f1-score   support

              BAS       0.88      0.75      0.81       110
              EOS       0.89      0.97      0.93       255
              HAC       0.82      0.81      0.81        93
              LYT       0.96      0.77      0.85       265
              MON       0.66      0.93      0.78       255
              NGB       0.75      0.76      0.75       235
              NGS       0.90      0.68      0.78       249

         accuracy                           0.82      1462
        macro avg       0.84      0.81      0.82      1462
     weighted avg       0.84      0.82      0.82      1462
```