```
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', 'Next LR', 'Monitor','% Improv', 'Duration')
      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), acc, loss)
    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}{monitor:^11s}{pimprov:^10.2f}{duration:^8.2f}'
    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_loss', 'V_acc', 'LR', 'Next LR', 'Monitor', '% Improv', 'I
                        print(msg)
                    except:
                        print('Invalid')
```

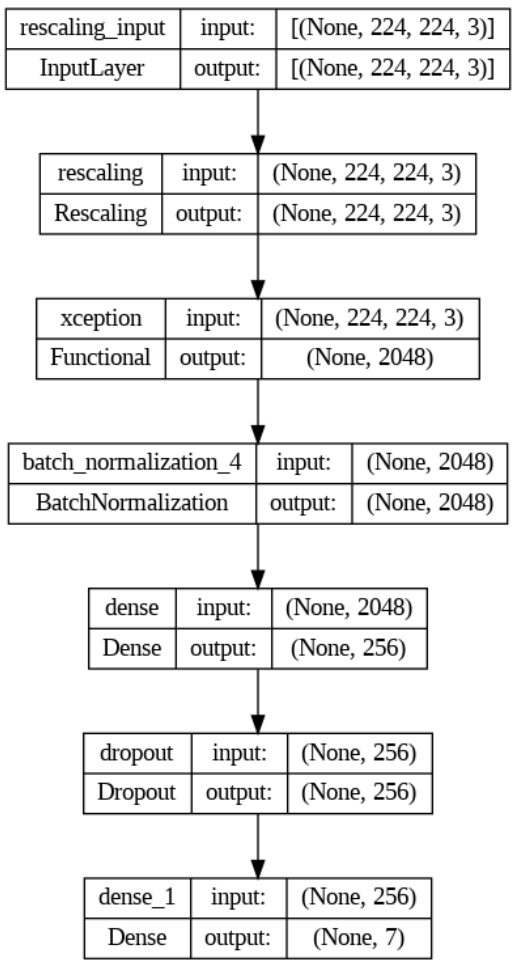
Xception

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


model = Sequential([
    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

 xception (Functional)       (None, 2048)              20861480

 batch_normalization_4 (Bat  (None, 2048)              8192
 chNormalization)

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 21396015 (81.62 MB)
Trainable params: 21337391 (81.40 MB)
Non-trainable params: 58624 (229.00 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)
```

```
 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: 0.1835s vs `on_train_batch_end` time: 0.2723s). Check your callbacks.
 1 /40     8.260   54.581   6.80736  63.406   0.00100  0.00100  accuracy     0.00    171.61
```

```
#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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-87b881e52cb0> in <cell line: 2>()
      1 #Create plots of the loss and accuracy on the training and validation sets:
----> 2 acc = history.history['accuracy']
      3 val_acc = history.history['val_accuracy']
      4
      5 loss = history.history['loss']

NameError: name 'history' is not defined
```

SEARCH STACK OVERFLOW

```
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 [==============================] - 9s 152ms/step
0.792749658002736
```

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

```
              precision    recall  f1-score   support

         BAS       0.81      0.57      0.67       110
         EOS       0.94      0.92      0.93       255
         HAC       0.68      0.74      0.71        93
         LYT       0.85      0.88      0.87       265
         MON       0.74      0.83      0.78       255
         NGB       0.73      0.65      0.69       235
         NGS       0.74      0.79      0.76       249

    accuracy                           0.79      1462
   macro avg       0.78      0.77      0.77      1462
weighted avg       0.79      0.79      0.79      1462
```