```
import pandas as pd
import seaborn as sns
import numpy as np
import os
import PIL
import pickle
from PIL import *
import cv2
import tensorflow as tf
import keras
from keras.applications import DenseNet121 # 2017 architecture
from keras.models import Model, load_model
from keras.initializers import glorot_uniform
from keras.utils import plot_model
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateScheduler
from IPython.display import display
from keras import *
from keras.preprocessing.image import ImageDataGenerator
from keras import layers, optimizers
from keras.applications.resnet50 import ResNet50
from keras.layers import *
from keras import backend as K
from keras import optimizers
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.regularizers import l2

```

```
from google.colab import drive
drive.mount('/content/drive')

# Load the dataset
facialexpression_df = pd.read_csv('/content/drive/MyDrive/DL Facial Recognition/icml_face_data.csv')
facialexpression_df = facialexpression_df.drop(columns=" Usage")
```

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

```
def conv_block(X, f, filters, stage, block, strides=(2, 2), weight_decay=1e-3):
    F1, F2, F3 = filters
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Save the input value for shortcut
    X_shortcut = X

    # First component of main path
    X = Conv2D(F1, (1, 1), strides=strides, padding='valid', name=conv_name_base + '2a',
               kernel_initializer=glorot_uniform(seed=0), kernel_regularizer=l2(weight_decay))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second component
    X = Conv2D(F2, (f, f), padding='same', name=conv_name_base + '2b',
               kernel_initializer=glorot_uniform(seed=0), kernel_regularizer=l2(weight_decay))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    # Third component
    X = Conv2D(F3, (1, 1), padding='valid', name=conv_name_base + '2c',
               kernel_initializer=glorot_uniform(seed=0), kernel_regularizer=l2(weight_decay))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    # Shortcut path
    X_shortcut = Conv2D(F3, (1, 1), strides=strides, padding='valid', name=conv_name_base + '1',
                        kernel_initializer=glorot_uniform(seed=0), kernel_regularizer=l2(weight_decay))(X_shortcut)
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X

def identity_block(X, f, filters, stage, block):
    """The identity block is the block that has no conv layer at shortcut."""
    F1, F2, F3 = filters
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    X_shortcut = X

    # First component of main path
    X = Conv2D(F1, (1, 1), padding='valid', name=conv_name_base + '2a', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    # Second component
    X = Conv2D(F2, (f, f), padding='same', name=conv_name_base + '2b', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    # Third component
    X = Conv2D(F3, (1, 1), padding='valid', name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    # Add shortcut
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X

def apply_blocks(X, filter_triplets, first_strides, num_identity_blocks, stage):
    """Apply a convolutional block followed by multiple identity blocks."""
    # Apply the first convolutional block with stride changes
    X = conv_block(X, 3, filter_triplets, stage, 'a', strides=first_strides)
    # Apply subsequent identity blocks
    for i in range(num_identity_blocks):
        X = identity_block(X, 3, filter_triplets, stage, chr(98 + i))  # 'b', 'c', etc.
    return X

```

```

# Define function to convert string to array
def string2array(x):
    return np.array(x.split(' ')).reshape(48, 48, 1).astype('float32')
# Define function to resize images
def resize(x):
    img = x.reshape(48, 48)
    return cv2.resize(img, dsize=(96, 96), interpolation=cv2.INTER_CUBIC)

```

```

# Apply preprocessing to pixels column
facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(lambda x: string2array(x))
facialexpression_df[' pixels'] =  facialexpression_df[' pixels'].apply(lambda x : resize(x))
# Define emotions of interest
emotions = [1]  # Example: you want to work with emotion label 1
label_to_text = {0: 'anger', 1 : 'disgust', 2 : 'fear', 3 : 'happiness', 4: 'sad', 5: 'surprise', 6: 'neutral'}

```

```
# Plot sample images
for i in emotions:
    data = facialexpression_df[facialexpression_df['emotion'] == i][:1]
    img = data[' pixels'].item()
    img = img.reshape(96, 96)

    plt.figure()
    plt.title(label_to_text[i])
    plt.imshow(img, cmap='gray')
    plt.axis('off')
```

```
from keras.utils import to_categorical

# Prepare input and target data
X = facialexpression_df[' pixels']
y = to_categorical(facialexpression_df['emotion'])

X = np.stack(X, axis = 0)
X = X.reshape(35887,96,96,1)

from sklearn.model_selection import train_test_split

# Split data into training, validation, and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, shuffle=True)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, shuffle=True)

X_train = X_train/255
X_val   = X_val /255
X_test  = X_test/255

# Data Augmentation
train_datagen = ImageDataGenerator(
rotation_range = 15,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.1,
    zoom_range = 0.1,
    horizontal_flip = True,
    fill_mode = "nearest")
```

```
# Define the input tensor shape
input_shape = (96, 96, 1)
X_input = Input(input_shape)

# Initial convolution and max pooling
X = ZeroPadding2D((3, 3))(X_input)
X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
X = BatchNormalization(axis=3, name='bn_conv1')(X)
X = Dropout(0.5)(X)  # Add dropout
X = Activation('relu')(X)
X = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(X)

# Helper function to apply a convolutional block and multiple identity blocks
def apply_blocks(X, filter_triplets, first_strides, num_identity_blocks, stage):
    # Apply convolutional block
    X = conv_block(X, 3, filter_triplets, stage, 'a', strides=first_strides)
    # Apply identity blocks
    for i in range(num_identity_blocks):
        X = identity_block(X, 3, filter_triplets, stage, chr(98 + i))  # 'b', 'c', ..., for identity blocks
    return X

# Stage 2
X = apply_blocks(X, [64, 64, 256], (1, 1), 2, 2)

# Stage 3
X = apply_blocks(X, [128, 128, 512], (2, 2), 3, 3)

# Stage 4
X = apply_blocks(X, [256, 256, 1024], (2, 2), 5, 4)

# Stage 5
X = apply_blocks(X, [512, 512, 2048], (2, 2), 2, 5)

# Global Average Pooling and final dense layer
X = GlobalAveragePooling2D()(X)
X = Dense(7, activation='softmax', name='final_output')(X)

# Create the model
model = Model(inputs=X_input, outputs=X, name='CustomResNet50')
# Show the model summary
model.summary()
```

```
 res5b_branch2b (Conv2D)     (None, 3, 3, 512)           2359808   ['activation_43[0][0]']

 bn5b_branch2b (BatchNormal  (None, 3, 3, 512)           2048      ['res5b_branch2b[0][0]']
 ization)

 activation_44 (Activation)  (None, 3, 3, 512)           0         ['bn5b_branch2b[0][0]']

 res5b_branch2c (Conv2D)     (None, 3, 3, 2048)          1050624   ['activation_44[0][0]']

 bn5b_branch2c (BatchNormal  (None, 3, 3, 2048)          8192      ['res5b_branch2c[0][0]']
 ization)

 add_14 (Add)                (None, 3, 3, 2048)          0         ['bn5b_branch2c[0][0]',
                                                                    'activation_42[0][0]']

 activation_45 (Activation)  (None, 3, 3, 2048)          0         ['add_14[0][0]']

 res5c_branch2a (Conv2D)     (None, 3, 3, 512)           1049088   ['activation_45[0][0]']

 bn5c_branch2a (BatchNormal  (None, 3, 3, 512)           2048      ['res5c_branch2a[0][0]']
 ization)

 activation_46 (Activation)  (None, 3, 3, 512)           0         ['bn5c_branch2a[0][0]']

 res5c_branch2b (Conv2D)     (None, 3, 3, 512)           2359808   ['activation_46[0][0]']

 bn5c_branch2b (BatchNormal  (None, 3, 3, 512)           2048      ['res5c_branch2b[0][0]']
 ization)

 activation_47 (Activation)  (None, 3, 3, 512)           0         ['bn5c_branch2b[0][0]']

 res5c_branch2c (Conv2D)     (None, 3, 3, 2048)          1050624   ['activation_47[0][0]']

 bn5c_branch2c (BatchNormal  (None, 3, 3, 2048)          8192      ['res5c_branch2c[0][0]']
 ization)

 add_15 (Add)                (None, 3, 3, 2048)          0         ['bn5c_branch2c[0][0]',
                                                                    'activation_45[0][0]']

 activation_48 (Activation)  (None, 3, 3, 2048)          0         ['add_15[0][0]']

 global_average_pooling2d (  (None, 2048)                0         ['activation_48[0][0]']
 GlobalAveragePooling2D)

 final_output (Dense)        (None, 7)                   14343     ['global_average_pooling2d[0][
                                                                    0]']

==================================================================================================
Total params: 23595783 (90.01 MB)
Trainable params: 23542663 (89.81 MB)
Non-trainable params: 53120 (207.50 KB)
__________________________________________________________________________________________________
```

```
# Define the total number of epochs and the number of warmup epochs
# Define the total number of epochs, the number of warmup epochs, and the batch size
total_epochs = 100
warmup_epochs = 5
batch_size = 64

# Define the initial learning rate and minimum learning rate
initial_learning_rate = 1e-3
min_learning_rate = 1e-6

# Calculate the total number of steps and warmup steps
total_steps = total_epochs * (len(X_train) // batch_size)
warmup_steps = warmup_epochs * (len(X_train) // batch_size)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, verbose=1, min_lr=1e-6)

# Define the learning rate scheduler with warmup
# Create the learning rate scheduler with warmup
# def lr_schedule(epoch, lr):
#     if epoch < warmup_epochs:
#         # Linear warmup
#         return initial_learning_rate * (epoch + 1) / warmup_epochs
#     else:
#         # Cosine annealing with restarts
#         cosine_decay = 0.5 * (1 + tf.cos(tf.constant(np.pi) * (epoch - warmup_epochs) / (total_epochs - warmup_epochs)))
#         decayed = (initial_learning_rate - min_learning_rate) * cosine_decay + min_learning_rate
#         return decayed
# using early stopping to exit training if validation loss is not decreasing even after certain epochs (patience)
earlystopping = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 20)

# save the best model with lower validation loss
checkpointer = ModelCheckpoint(filepath = "/content/drive/MyDrive/DL Facial Recognition /FacialExpression_weights_resnet.hdf5", verbose =
# Compile the model
model.compile(keras.optimizers.Adam(lr = 1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)


```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.L
Model: "CustomResNet50"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 96, 96, 1)]       0         []

 zero_padding2d (ZeroPaddin  (None, 102, 102, 1)       0         ['input_1[0][0]']
 g2D)

 conv1 (Conv2D)              (None, 48, 48, 64)        3200      ['zero_padding2d[0][0]']

 bn_conv1 (BatchNormalizati  (None, 48, 48, 64)        256       ['conv1[0][0]']
 on)

 dropout (Dropout)           (None, 48, 48, 64)        0         ['bn_conv1[0][0]']

 activation (Activation)     (None, 48, 48, 64)        0         ['dropout[0][0]']

 max_pooling2d (MaxPooling2  (None, 24, 24, 64)        0         ['activation[0][0]']
 D)

 res2a_branch2a (Conv2D)     (None, 24, 24, 64)        4160      ['max_pooling2d[0][0]']

 bn2a_branch2a (BatchNormal  (None, 24, 24, 64)        256       ['res2a_branch2a[0][0]']
 ization)

 activation_1 (Activation)   (None, 24, 24, 64)        0         ['bn2a_branch2a[0][0]']

 res2a_branch2b (Conv2D)     (None, 24, 24, 64)        36928     ['activation_1[0][0]']

 bn2a_branch2b (BatchNormal  (None, 24, 24, 64)        256       ['res2a_branch2b[0][0]']
 ization)

 activation_2 (Activation)   (None, 24, 24, 64)        0         ['bn2a_branch2b[0][0]']

 res2a_branch2c (Conv2D)     (None, 24, 24, 256)       16640     ['activation_2[0][0]']

 res2a_branch1 (Conv2D)      (None, 24, 24, 256)       16640     ['max_pooling2d[0][0]']

 bn2a_branch2c (BatchNormal  (None, 24, 24, 256)       1024      ['res2a_branch2c[0][0]']
 ization)
```

```
bn2a_branch1 (BatchNormali  (None, 24, 24, 256)          1024      ['res2a_branch1[0][0]']
zation)

add (Add)                   (None, 24, 24, 256)          0         ['bn2a_branch2c[0][0]',
                                                                    'bn2a_branch1[0][0]']

activation_3 (Activation)   (None, 24, 24, 256)          0         ['add[0][0]']

res2b_branch2a (Conv2D)     (None, 24, 24, 64)           16448     ['activation_3[0][0]']

bn2b_branch2a (BatchNormal  (None, 24, 24, 64)           256       ['res2b_branch2a[0][0]']
ization)

activation_4 (Activation)   (None, 24, 24, 64)           0         ['bn2b_branch2a[0][0]']
```

```
history = model.fit(train_generator, steps_per_epoch=len(X_train)//batch_size, epochs=total_epochs, validation_data=(X_val, y_val), valid
```

```
504/504 [==============================] - ETA: 0s - loss: 1.0369 - accuracy: 0.6308
Epoch 39: val_loss improved from 1.12797 to 1.10169, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 42s 84ms/step - loss: 1.0369 - accuracy: 0.6308 - val_loss: 1.1017 - val_accuracy: 0.6003 -
Epoch 40/100
504/504 [==============================] - ETA: 0s - loss: 1.0262 - accuracy: 0.6326
Epoch 40: val_loss did not improve from 1.10169
504/504 [==============================] - 38s 74ms/step - loss: 1.0262 - accuracy: 0.6326 - val_loss: 1.1685 - val_accuracy: 0.5808 -
Epoch 41/100
504/504 [==============================] - ETA: 0s - loss: 1.0206 - accuracy: 0.6362
Epoch 41: val_loss did not improve from 1.10169
504/504 [==============================] - 37s 74ms/step - loss: 1.0206 - accuracy: 0.6362 - val_loss: 1.1742 - val_accuracy: 0.5747 -
Epoch 42/100
504/504 [==============================] - ETA: 0s - loss: 1.0144 - accuracy: 0.6385
Epoch 42: val_loss did not improve from 1.10169
504/504 [==============================] - 37s 74ms/step - loss: 1.0144 - accuracy: 0.6385 - val_loss: 1.1298 - val_accuracy: 0.5942 -
Epoch 43/100
504/504 [==============================] - ETA: 0s - loss: 1.0081 - accuracy: 0.6395
Epoch 43: val_loss improved from 1.10169 to 1.09355, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 42s 84ms/step - loss: 1.0081 - accuracy: 0.6395 - val_loss: 1.0935 - val_accuracy: 0.5987 -
Epoch 44/100
504/504 [==============================] - ETA: 0s - loss: 0.9997 - accuracy: 0.6444
Epoch 44: val_loss did not improve from 1.09355
504/504 [==============================] - 38s 75ms/step - loss: 0.9997 - accuracy: 0.6444 - val_loss: 1.1084 - val_accuracy: 0.6048 -
Epoch 45/100
504/504 [==============================] - ETA: 0s - loss: 0.9912 - accuracy: 0.6430
Epoch 45: val_loss improved from 1.09355 to 1.09098, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 42s 84ms/step - loss: 0.9912 - accuracy: 0.6430 - val_loss: 1.0910 - val_accuracy: 0.6120 -
Epoch 46/100
504/504 [==============================] - ETA: 0s - loss: 0.9792 - accuracy: 0.6460
Epoch 46: val_loss did not improve from 1.09098
504/504 [==============================] - 38s 74ms/step - loss: 0.9792 - accuracy: 0.6460 - val_loss: 1.1025 - val_accuracy: 0.6148 -
Epoch 47/100
504/504 [==============================] - ETA: 0s - loss: 0.9772 - accuracy: 0.6478
Epoch 47: val_loss did not improve from 1.09098
504/504 [==============================] - 37s 74ms/step - loss: 0.9772 - accuracy: 0.6478 - val_loss: 1.1394 - val_accuracy: 0.5875 -
Epoch 48/100
504/504 [==============================] - ETA: 0s - loss: 0.9735 - accuracy: 0.6511
Epoch 48: val_loss did not improve from 1.09098
504/504 [==============================] - 37s 74ms/step - loss: 0.9735 - accuracy: 0.6511 - val_loss: 1.1099 - val_accuracy: 0.6070 -
Epoch 49/100
504/504 [==============================] - ETA: 0s - loss: 0.9638 - accuracy: 0.6520
Epoch 49: val_loss did not improve from 1.09098
504/504 [==============================] - 37s 74ms/step - loss: 0.9638 - accuracy: 0.6520 - val_loss: 1.0957 - val_accuracy: 0.6031 -
Epoch 50/100
504/504 [==============================] - ETA: 0s - loss: 0.9596 - accuracy: 0.6564
Epoch 50: val_loss did not improve from 1.09098
504/504 [==============================] - 37s 74ms/step - loss: 0.9596 - accuracy: 0.6564 - val_loss: 1.1052 - val_accuracy: 0.6020 -
Epoch 51/100
504/504 [==============================] - ETA: 0s - loss: 0.9547 - accuracy: 0.6557
Epoch 51: val_loss did not improve from 1.09098
504/504 [==============================] - 37s 74ms/step - loss: 0.9547 - accuracy: 0.6557 - val_loss: 1.0956 - val_accuracy: 0.5964 -
Epoch 52/100
504/504 [==============================] - ETA: 0s - loss: 0.9477 - accuracy: 0.6586
Epoch 52: val_loss improved from 1.09098 to 1.08075, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 42s 84ms/step - loss: 0.9477 - accuracy: 0.6586 - val_loss: 1.0808 - val_accuracy: 0.6065 -
Epoch 53/100
504/504 [==============================] - ETA: 0s - loss: 0.9401 - accuracy: 0.6610
Epoch 53: val_loss did not improve from 1.08075
```

```
score = model.evaluate(X_test, y_test)
print('Test Accuracy: {}'.format(score[1]))
```

```
57/57 [==============================] - 2s 18ms/step - loss: 0.9468 - accuracy: 0.6546
Test Accuracy: 0.6545960903167725
```

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```