```
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
```

```
# RESBLOCK :=> input -> Convolution Block -> Identity Block -> Identity Block -> output
def res_block(X, filters,stage):
    # Convolution Block
    X_copy = X

    f1, f2, f3 = filters

    # Main path
    X = Conv2D(f1,(1,1), strides = (1,1), name = 'res_'+str(stage)+'_conv_a', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = MaxPool2D((2,2))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_conv_a')(X)
    X = Activation('relu')(X)

    X = Conv2D(f2,(3,3), strides = (1,1), padding= 'same', name = 'res_' + str(stage) + '_conv_b', kernel_initializer = glorot_uniform(se

    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_conv_b')(X)
    X = Activation('relu')(X)

    X = Conv2D(f3,(1,1),strides = (1,1), name ='res_' + str(stage) + '_conv_c', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_conv_c')(X)

    # Short path

    X_copy = Conv2D(f3,(1,1), strides =(1,1), name = 'res_'+str(stage) +'_conv_copy', kernel_initializer = glorot_uniform(seed = 0))(X_co
    X_copy = MaxPool2D((2,2))(X_copy)
    X_copy = BatchNormalization(axis = 3, name='bn_'+str(stage)+'_conv_copy')(X_copy)

    # ADD
    X = Add()([X,X_copy])
    X = Activation('relu')(X)

    # Identity BLock 1
    X_copy = X

    # main path

    X = Conv2D(f1,(1,1), strides = (1,1), name = 'res_'+str(stage)+'_identity_1_a', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_1_a')(X)
    X = Activation('relu')(X)

    X = Conv2D(f2,(3,3), strides = (1,1), padding= 'same', name = 'res_' + str(stage) + '_identity_1_b', kernel_initializer = glorot_unif
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_1_b')(X)
    X = Activation('relu')(X)

    X = Conv2D(f3, (1,1),strides = (1,1), name ='res_' + str(stage) + '_identity_1_c', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_1_c')(X)

    # ADD
    X = Add()([X,X_copy])
    X = Activation('relu')(X)

    # identity block 2
    X_copy = X

    # main path
    X = Conv2D(f1,(1,1), strides = (1,1), name = 'res_'+str(stage)+'_identity_2_a', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_2_a')(X)
    X = Activation('relu')(X)

    X = Conv2D(f2,(3,3), strides = (1,1), padding= 'same', name = 'res_' + str(stage) + '_identity_2_b', kernel_initializer = glorot_unif
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_2_b')(X)
    X = Activation('relu')(X)

    X = Conv2D(f3, (1,1),strides = (1,1), name ='res_' + str(stage) + '_identity_2_c', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_'+ str(stage) + '_identity_2_c')(X)
    # ADD
    X = Add()([X,X_copy])
    X = Activation('relu')(X)

    return X
```

```
from google.colab import drive
drive.mount('/content/drive')
facialexpression_df = pd.read_csv('/content/drive/MyDrive/DL Facial Recognition/icml_face_data.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
facialexpression_df = facialexpression_df.drop(columns = " Usage")

```

```
facialexpression_df.head()
```

	emotion	pixels
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	231 212 156 164 174 138 161 173 182 200 106 38...
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...

```
def string2array(x):
    return np.array(x.split(' ')).reshape(48,48,1).astype('float32')
```

```
Start coding or generate with AI.
```

```
def resize(x):
    img = x.reshape(48,48)
    return cv2.resize(img, dsize=(96,96), interpolation = cv2.INTER_CUBIC)
```

```
facialexpression_df[' pixels'] = facialexpression_df[' pixels'].apply(lambda x : string2array(x))
facialexpression_df[' pixels'] =  facialexpression_df[' pixels'].apply(lambda x : resize(x))
```

```
facialexpression_df.shape
```

```
(35887, 2)
```

```
label_to_text = {0: 'anger', 1 : 'disgust', 2 : 'fear', 3 : 'happiness', 4: 'sad', 5: 'surprise', 6: 'neutral'}
```

```
emotions = [1]
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')
```


```
facialexpression_df.emotion.value_counts().index
```

```
Index([3, 6, 4, 2, 0, 5, 1], dtype='int64', name='emotion')
```

```
facialexpression_df.emotion.value_counts()
```

```
emotion
3    8989
6    6198
4    6077
2    5121
0    4953
5    4002
1     547
Name: count, dtype: int64
```

```
class1 = facialexpression_df.loc[facialexpression_df['emotion'] == 1]
class1.shape
```

```
(547, 2)
```

```
from keras.utils import to_categorical

X = facialexpression_df[' pixels']
y = to_categorical(facialexpression_df['emotion'])

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

print(X.shape, y.shape)
```

```
(35887, 96, 96, 1) (35887, 7)
```

```
from sklearn.model_selection import train_test_split
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.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((32298, 96, 96, 1), (1795, 96, 96, 1), (32298, 7), (1795, 7))
```

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

```
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")
```

```
input_shape = (96, 96, 1)

# Input tensor shape
X_input = Input(input_shape)

# Zero-padding
X = ZeroPadding2D((3, 3))(X_input)

# 1 - stage
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 = Activation('relu')(X)
X = MaxPooling2D((3, 3), strides= (2, 2))(X)

# 2 - stage
X = res_block(X, [64, 64, 256], stage= 2)

# 3 - stage
X = res_block(X, [128, 128, 512], stage= 3)

# 4 - stage
# X = res_block(X, filter= [256, 256, 1024], stage= 4)

# Average Pooling
X = AveragePooling2D((4, 4), name = 'Averagea_Pooling')(X)

# Final layer
X = Flatten()(X)
X = Dense(7, activation = 'softmax', name = 'Dense_final', kernel_initializer= glorot_uniform(seed=0))(X)

facialmodel = Model(inputs= X_input, outputs = X, name = 'Resnet18')

facialmodel.summary()
```

```
Model: "Resnet18"
_________________________________________________________________________________________________
 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)

 activation (Activation)     (None, 48, 48, 64)        0         ['bn_conv1[0][0]']

 max_pooling2d (MaxPooling2  (None, 23, 23, 64)        0         ['activation[0][0]']
 D)

 res_2_conv_a (Conv2D)       (None, 23, 23, 64)        4160      ['max_pooling2d[0][0]']

 max_pooling2d_1 (MaxPoolin  (None, 11, 11, 64)        0         ['res_2_conv_a[0][0]']
 g2D)

 bn_2_conv_a (BatchNormaliz  (None, 11, 11, 64)        256       ['max_pooling2d_1[0][0]']
 ation)

 activation_1 (Activation)   (None, 11, 11, 64)        0         ['bn_2_conv_a[0][0]']

 res_2_conv_b (Conv2D)       (None, 11, 11, 64)        36928     ['activation_1[0][0]']

 bn_2_conv_b (BatchNormaliz  (None, 11, 11, 64)        256       ['res_2_conv_b[0][0]']
 ation)

 activation_2 (Activation)   (None, 11, 11, 64)        0         ['bn_2_conv_b[0][0]']

 res_2_conv_copy (Conv2D)    (None, 23, 23, 256)       16640     ['max_pooling2d[0][0]']

 res_2_conv_c (Conv2D)       (None, 11, 11, 256)       16640     ['activation_2[0][0]']

 max_pooling2d_2 (MaxPoolin  (None, 11, 11, 256)       0         ['res_2_conv_copy[0][0]']
 g2D)

 bn_2_conv_c (BatchNormaliz  (None, 11, 11, 256)       1024      ['res_2_conv_c[0][0]']
 ation)

 bn_2_conv_copy (BatchNorma  (None, 11, 11, 256)       1024      ['max_pooling2d_2[0][0]']
```

```
lization)

add (Add)                   (None, 11, 11, 256)          0         ['bn_2_conv_c[0][0]',
                                                                    'bn_2_conv_copy[0][0]']

activation_3 (Activation)   (None, 11, 11, 256)          0         ['add[0][0]']

res_2_identity_1_a (Conv2D  (None, 11, 11, 64)           16448     ['activation_3[0][0]']
)

bn_2_identity_1_a (BatchNo  (None, 11, 11, 64)           256       ['res_2_identity_1_a[0][0]']
```

```
facialmodel.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3), loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
# 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.hdf5", verbose = 1, sav
```

```
history = facialmodel.fit(train_datagen.flow(X_train, y_train, batch_size=64),
    validation_data= (X_val, y_val), steps_per_epoch=len(X_train) // 64,
    epochs= 100, callbacks=[checkpointer, earlystopping])
```

```
504/504 [==============================] - ETA: 0s - loss: 0.7907 - accuracy: 0.7022
Epoch 27: val_loss did not improve from 0.98270
504/504 [==============================] - 29s 58ms/step - loss: 0.7907 - accuracy: 0.7022 - val_loss: 0.9974 - val_accuracy: 0.6282
Epoch 28/100
504/504 [==============================] - ETA: 0s - loss: 0.7845 - accuracy: 0.7062
Epoch 28: val_loss did not improve from 0.98270
504/504 [==============================] - 29s 57ms/step - loss: 0.7845 - accuracy: 0.7062 - val_loss: 1.0917 - val_accuracy: 0.5948
Epoch 29/100
504/504 [==============================] - ETA: 0s - loss: 0.7756 - accuracy: 0.7090
Epoch 29: val_loss improved from 0.98270 to 0.97295, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 30s 59ms/step - loss: 0.7756 - accuracy: 0.7090 - val_loss: 0.9729 - val_accuracy: 0.6382
Epoch 30/100
504/504 [==============================] - ETA: 0s - loss: 0.7640 - accuracy: 0.7163
Epoch 30: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7640 - accuracy: 0.7163 - val_loss: 1.2254 - val_accuracy: 0.5368
Epoch 31/100
504/504 [==============================] - ETA: 0s - loss: 0.7582 - accuracy: 0.7155
Epoch 31: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7582 - accuracy: 0.7155 - val_loss: 1.0067 - val_accuracy: 0.6249
Epoch 32/100
504/504 [==============================] - ETA: 0s - loss: 0.7490 - accuracy: 0.7198
Epoch 32: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7490 - accuracy: 0.7198 - val_loss: 1.0215 - val_accuracy: 0.6137
Epoch 33/100
504/504 [==============================] - ETA: 0s - loss: 0.7392 - accuracy: 0.7228
Epoch 33: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7392 - accuracy: 0.7228 - val_loss: 0.9828 - val_accuracy: 0.6349
Epoch 34/100
504/504 [==============================] - ETA: 0s - loss: 0.7309 - accuracy: 0.7254
Epoch 34: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 58ms/step - loss: 0.7309 - accuracy: 0.7254 - val_loss: 1.1056 - val_accuracy: 0.6193
Epoch 35/100
504/504 [==============================] - ETA: 0s - loss: 0.7267 - accuracy: 0.7284
Epoch 35: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7267 - accuracy: 0.7284 - val_loss: 1.1731 - val_accuracy: 0.5680
Epoch 36/100
504/504 [==============================] - ETA: 0s - loss: 0.7145 - accuracy: 0.7351
Epoch 36: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7145 - accuracy: 0.7351 - val_loss: 1.6655 - val_accuracy: 0.4805
Epoch 37/100
504/504 [==============================] - ETA: 0s - loss: 0.7083 - accuracy: 0.7360
Epoch 37: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.7083 - accuracy: 0.7360 - val_loss: 1.0107 - val_accuracy: 0.6343
Epoch 38/100
504/504 [==============================] - ETA: 0s - loss: 0.6990 - accuracy: 0.7375
Epoch 38: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 58ms/step - loss: 0.6990 - accuracy: 0.7375 - val_loss: 0.9897 - val_accuracy: 0.6399
Epoch 39/100
504/504 [==============================] - ETA: 0s - loss: 0.6908 - accuracy: 0.7412
Epoch 39: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 57ms/step - loss: 0.6908 - accuracy: 0.7412 - val_loss: 1.0663 - val_accuracy: 0.6332
Epoch 40/100
504/504 [==============================] - ETA: 0s - loss: 0.6807 - accuracy: 0.7446
Epoch 40: val_loss did not improve from 0.97295
504/504 [==============================] - 29s 58ms/step - loss: 0.6807 - accuracy: 0.7446 - val_loss: 1.0874 - val_accuracy: 0.6221
Epoch 41/100
504/504 [==============================] - ETA: 0s - loss: 0.6735 - accuracy: 0.7472
Epoch 41: val_loss did not improve from 0.97295
```

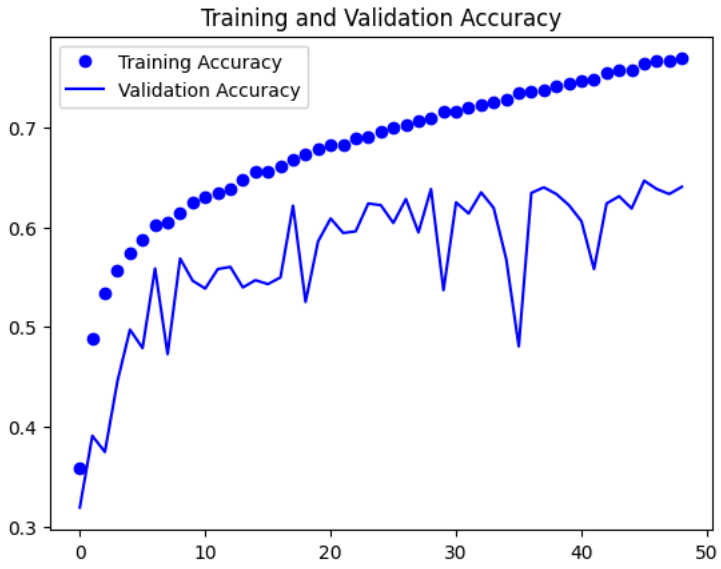
```
score = facialmodel.evaluate(X_test, y_test)
print('Test Accuracy: {}'.format(score[1]))
```

```
57/57 [==============================] - 0s 5ms/step - loss: 0.9472 - accuracy: 0.6819
Test Accuracy: 0.6818941235542297
```

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

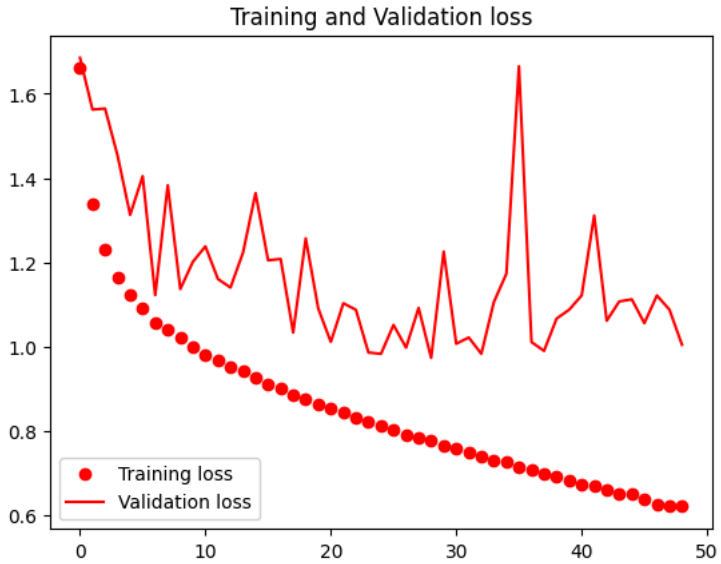
```
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b0eaad13040>
```




```
plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b0e241be620>
```

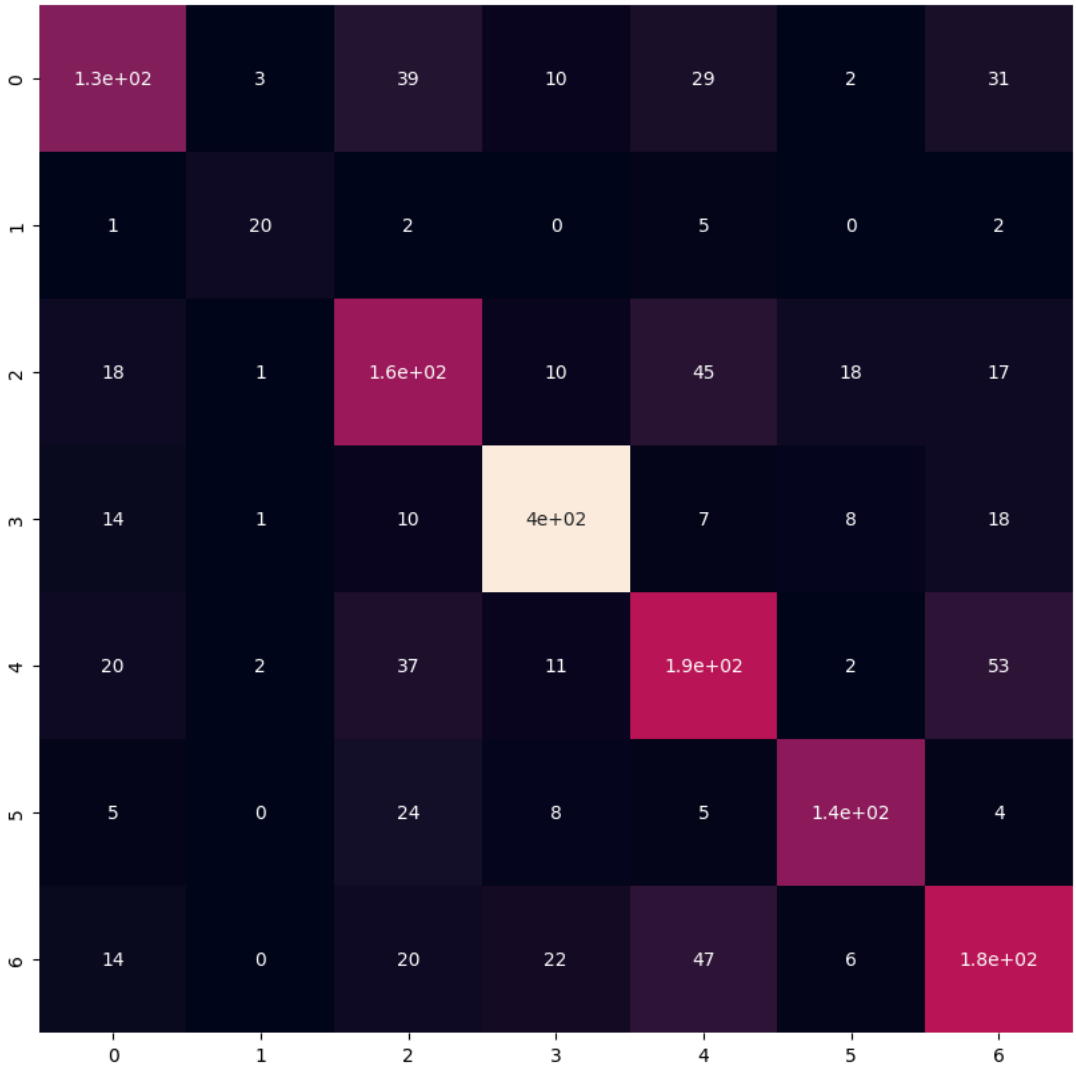


```
predicted_classes = np.argmax(facialmodel.predict(X_test), axis=-1)
y_true = np.argmax(y_test, axis=-1)
```

```
57/57 [==============================] - 1s 3ms/step
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, predicted_classes)
plt.figure(figsize = (10, 10))
sns.heatmap(cm, annot = True, cbar = False)
```

```
<Axes: >
```




```
L = 5
W = 5

fig, axes = plt.subplots(L, W, figsize = (24, 24))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i].reshape(96,96), cmap = 'gray')
    axes[i].set_title('Prediction = {}\n True = {}'.format(label_to_text[predicted_classes[i]], label_to_text[y_true[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace = 1)
```


Prediction = happiness
True = happiness
Prediction = anger
True = anger
Prediction = happiness
True = fear
Prediction = neutral
True = neutral
Prediction = happiness
True = happiness
Prediction = neutral
True = fear
Prediction = sad
True = neutral
Prediction = sad
True = sad
Prediction = sad
True = sad
Prediction = neutral
True = neutral
Prediction = anger
True = anger
Prediction = neutral
True = neutral
Prediction = fear
True = sad
Prediction = surprise
True = surprise
Prediction = fear
True = fear
Prediction = fear
True = fear
Prediction = surprise
True = surprise
Prediction = happiness
True = happiness
Prediction = surprise
True = neutral
Prediction = surprise
True = surprise
Prediction = anger
True = anger
Prediction = happiness
True = fear
Prediction = happiness
True = happiness
Prediction = happiness
True = happiness
Prediction = happiness
True = happiness

```
Start coding or generate with AI.
```

```
# Compute accuracy
accuracy = np.mean(predicted_classes == y_true)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.6818941504178273
```

```
def identity_block(X, f, filters, stage, block):
    """
    Implementation of the identity block as defined in Figure 3

    Arguments:
    X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
    f -- integer, specifying the shape of the middle CONV's window for the main path
    filters -- python list of integers, defining the number of filters in the CONV layers of the main path
    stage -- integer, used to name the layers, depending on their position in the network
    block -- string/character, used to name the layers, depending on their position in the network

    Returns:
    X -- output of the identity block, tensor of shape (n_H, n_W, n_C)
    """

    # defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value. You'll need this later to add back to the main path.
    X_shortcut = X

    # First component of main path
    X = Conv2D(F1, (1, 1), strides=(1, 1), 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 of main path (≈3 lines)
    X = Conv2D(F2, (f, f), strides=(1, 1), 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 of main path (≈2 lines)
    X = Conv2D(F3, (1, 1), strides=(1, 1), name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    # Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X

def convolutional_block(X, f, filters, stage, block, s=2):
    """
    Implementation of the convolutional block as defined in Figure 4

    Arguments:
    X -- input tensor of shape (m, n_H_prev, n_W_prev, n_C_prev)
    f -- integer, specifying the shape of the middle CONV's window for the main path
    filters -- python list of integers, defining the number of filters in the CONV layers of the main path
    stage -- integer, used to name the layers, depending on their position in the network
    block -- string/character, used to name the layers, depending on their position in the network
    s -- Integer, specifying the stride to be used

    Returns:
    X -- output of the convolutional block, tensor of shape (n_H, n_W, n_C)
    """

    # defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    # Retrieve Filters
    F1, F2, F3 = filters

    # Save the input value
    X_shortcut = X

    # First component of main path
    X = Conv2D(F1, (1, 1), strides=(s, s), 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 of main path
```

```
    X = Conv2D(F2, (f, f), strides=(1, 1), 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 of main path
    X = Conv2D(F3, (1, 1), strides=(1, 1), name=conv_name_base + '2c', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

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

    # Add shortcut value to main path
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X

def ResNet50(input_shape=(96, 96, 1), classes=7):
    """
    Implementation of the popular ResNet50 the following architecture:
    CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK*2 -> CONVBLOCK -> IDBLOCK*3
    -> CONVBLOCK -> IDBLOCK*5 -> CONVBLOCK -> IDBLOCK*2 -> AVGPOOL -> TOPLAYER

    Arguments:
    input_shape -- shape of the images of the dataset
    classes -- integer, number of classes

    Returns:
    model -- a Model() instance in Keras
    """

    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)

    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    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 = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

    # Stage 3
    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Stage 4
    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Stage 5
    X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    # AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)

    # output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer=glorot_uniform(seed=0))(X)

    # Create model
    model = Model(inputs=X_input, outputs=X, name='ResNet50')
```

```
    return model
# Create ResNet50 model
facial_model = ResNet50(input_shape=(96, 96, 1), classes=7)

# Print model summary
facial_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 1)]       0

 block1_conv1 (Conv2D)       (None, 96, 96, 64)        640

 block1_conv2 (Conv2D)       (None, 96, 96, 64)        36928

 block1_pool (MaxPooling2D)  (None, 48, 48, 64)        0

 block2_conv1 (Conv2D)       (None, 48, 48, 128)       73856

 block2_conv2 (Conv2D)       (None, 48, 48, 128)       147584

 block2_pool (MaxPooling2D)  (None, 24, 24, 128)       0

 block3_conv1 (Conv2D)       (None, 24, 24, 256)       295168

 block3_conv2 (Conv2D)       (None, 24, 24, 256)       590080

 block3_conv3 (Conv2D)       (None, 24, 24, 256)       590080

 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0

 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160

 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0

 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 flatten (Flatten)           (None, 4608)              0

 fc1 (Dense)                 (None, 4096)              18878464

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 7)                 28679

=================================================================
Total params: 50401991 (192.27 MB)
Trainable params: 50401991 (192.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```


# Compile the model
facial_model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3), loss = "categorical_crossentropy", metrics = ["accuracy"])
```

```
# 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_VGG.hdf5", verbose = 1,
```

```
history = facial_model.fit(train_datagen.flow(X_train, y_train, batch_size=64),
    validation_data= (X_val, y_val), steps_per_epoch=len(X_train) // 64,
    epochs= 100, callbacks=[checkpointer, earlystopping])
```

```
Epoch 1/100
504/504 [==============================] - ETA: 0s - loss: 1.8179 - accuracy: 0.2489
Epoch 1: val_loss improved from inf to 1.80318, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_weights
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via
  saving_api.save_model(
504/504 [==============================] - 40s 65ms/step - loss: 1.8179 - accuracy: 0.2489 - val_loss: 1.8032 - val_accuracy: 0.2469
Epoch 2/100
504/504 [==============================] - ETA: 0s - loss: 1.8131 - accuracy: 0.2504
Epoch 2: val_loss improved from 1.80318 to 1.80172, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_wei
504/504 [==============================] - 36s 71ms/step - loss: 1.8131 - accuracy: 0.2504 - val_loss: 1.8017 - val_accuracy: 0.2469
Epoch 3/100
504/504 [==============================] - ETA: 0s - loss: 1.8127 - accuracy: 0.2502
Epoch 3: val_loss improved from 1.80172 to 1.80084, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_wei
504/504 [==============================] - 36s 70ms/step - loss: 1.8127 - accuracy: 0.2502 - val_loss: 1.8008 - val_accuracy: 0.2469
Epoch 4/100
504/504 [==============================] - ETA: 0s - loss: 1.8128 - accuracy: 0.2504
Epoch 4: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 58ms/step - loss: 1.8128 - accuracy: 0.2504 - val_loss: 1.8027 - val_accuracy: 0.2469
Epoch 5/100
504/504 [==============================] - ETA: 0s - loss: 1.8125 - accuracy: 0.2504
Epoch 5: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 57ms/step - loss: 1.8125 - accuracy: 0.2504 - val_loss: 1.8039 - val_accuracy: 0.2469
Epoch 6/100
504/504 [==============================] - ETA: 0s - loss: 1.8125 - accuracy: 0.2504
Epoch 6: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 58ms/step - loss: 1.8125 - accuracy: 0.2504 - val_loss: 1.8016 - val_accuracy: 0.2469
Epoch 7/100
504/504 [==============================] - ETA: 0s - loss: 1.8126 - accuracy: 0.2504
Epoch 7: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 58ms/step - loss: 1.8126 - accuracy: 0.2504 - val_loss: 1.8017 - val_accuracy: 0.2469
Epoch 8/100
504/504 [==============================] - ETA: 0s - loss: 1.8123 - accuracy: 0.2505
Epoch 8: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 57ms/step - loss: 1.8123 - accuracy: 0.2505 - val_loss: 1.8022 - val_accuracy: 0.2469
Epoch 9/100
504/504 [==============================] - ETA: 0s - loss: 1.8121 - accuracy: 0.2504
Epoch 9: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 58ms/step - loss: 1.8121 - accuracy: 0.2504 - val_loss: 1.8013 - val_accuracy: 0.2469
Epoch 10/100
504/504 [==============================] - ETA: 0s - loss: 1.8120 - accuracy: 0.2504
Epoch 10: val_loss did not improve from 1.80084
504/504 [==============================] - 29s 58ms/step - loss: 1.8120 - accuracy: 0.2504 - val_loss: 1.8012 - val_accuracy: 0.2469
Epoch 11/100
504/504 [==============================] - ETA: 0s - loss: 1.8120 - accuracy: 0.2504
Epoch 11: val_loss improved from 1.80084 to 1.80050, saving model to /content/drive/MyDrive/DL Facial Recognition /FacialExpression_we
504/504 [==============================] - 36s 71ms/step - loss: 1.8120 - accuracy: 0.2504 - val_loss: 1.8005 - val_accuracy: 0.2469
Epoch 12/100
504/504 [==============================] - ETA: 0s - loss: 1.8118 - accuracy: 0.2506
Epoch 12: val_loss did not improve from 1.80050
504/504 [==============================] - 29s 58ms/step - loss: 1.8118 - accuracy: 0.2506 - val_loss: 1.8008 - val_accuracy: 0.2469
Epoch 13/100
504/504 [==============================] - ETA: 0s - loss: 1.8118 - accuracy: 0.2504
Epoch 13: val_loss did not improve from 1.80050
504/504 [==============================] - 29s 57ms/step - loss: 1.8118 - accuracy: 0.2504 - val_loss: 1.8018 - val_accuracy: 0.2469
Epoch 14/100
504/504 [==============================] - ETA: 0s - loss: 1.8119 - accuracy: 0.2503
Epoch 14: val_loss did not improve from 1.80050
```

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

```
57/57 [==============================] - 1s 12ms/step - loss: 1.8113 - accuracy: 0.2552
Test Accuracy: 0.25515320897102356
```

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

```
epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b0d40dc3700>
```