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

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

```
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...

Next steps: View recommended plots

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

```
def resize(x):
    img = x.reshape(48,48)
    temp = cv2.resize(img, dsize=(48,48), interpolation = cv2.INTER_CUBIC)
    return np.stack([temp,temp,temp], axis = 2)
```

```
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'}
```

```
facialexpression_df[' pixels'][1].shape
```

```
(48, 48, 3)
```

```
emotions = [1]
for i in emotions:
    data = facialexpression_df[facialexpression_df['emotion']==i][:1]

    img = data[' pixels'].item()
    img = img.reshape(48,48, 3)

    plt.figure()
    plt.title(label_to_text[i])
    plt.imshow(img/255)
    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(35887,48,48,3)

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

```
(35887, 48, 48, 3) (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, 48, 48, 3), (1795, 48, 48, 3), (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")
```

```
from keras.applications import VGG16

input_shape = (48,48, 3)

base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

VGGmodel = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(7, activation='sigmoid')
])

VGGmodel.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688  
                                                                 
 flatten (Flatten)           (None, 512)               0         
                                                                 
 dense (Dense)               (None, 256)               131328    
                                                                 
 dense_1 (Dense)             (None, 7)                 1799      
                                                                 
=================================================================
Total params: 14847815 (56.64 MB)
Trainable params: 14847815 (56.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
VGGmodel.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 /VGGFacialExpression_weights.hdf5", verbose = 1, sav
```

```
history = VGGmodel.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])
```

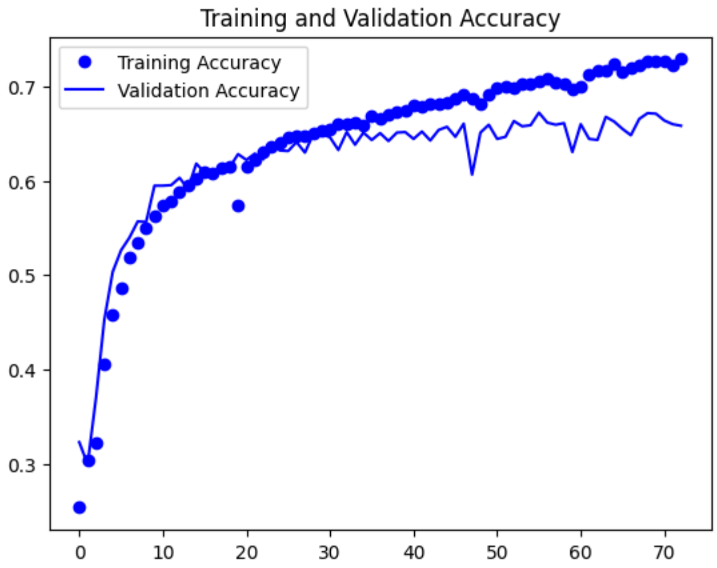
```
Epoch 57/100
504/504 [==============================] - ETA: 0s - loss: 0.7979 - accuracy: 0.7090
Epoch 57: val_loss did not improve from 0.92941
504/504 [==============================] - 45s 90ms/step - loss: 0.7979 - accuracy: 0.7090 - val_loss: 0.9443 - val_accuracy: 0.6616
Epoch 58/100
504/504 [==============================] - ETA: 0s - loss: 0.8075 - accuracy: 0.7034
Epoch 58: val_loss did not improve from 0.92941
504/504 [==============================] - 45s 88ms/step - loss: 0.8075 - accuracy: 0.7034 - val_loss: 0.9666 - val_accuracy: 0.6594
Epoch 59/100
504/504 [==============================] - ETA: 0s - loss: 0.8156 - accuracy: 0.7025
Epoch 59: val_loss did not improve from 0.92941
504/504 [==============================] - 44s 87ms/step - loss: 0.8156 - accuracy: 0.7025 - val_loss: 0.9393 - val_accuracy: 0.6611
Epoch 60/100
504/504 [==============================] - ETA: 0s - loss: 0.8361 - accuracy: 0.6968
Epoch 60: val_loss did not improve from 0.92941
504/504 [==============================] - 45s 90ms/step - loss: 0.8361 - accuracy: 0.6968 - val_loss: 1.0100 - val_accuracy: 0.6304
Epoch 61/100
504/504 [==============================] - ETA: 0s - loss: 0.8257 - accuracy: 0.6992
Epoch 61: val_loss did not improve from 0.92941
504/504 [==============================] - 43s 86ms/step - loss: 0.8257 - accuracy: 0.6992 - val_loss: 0.9997 - val_accuracy: 0.6600
Epoch 62/100
504/504 [==============================] - ETA: 0s - loss: 0.7871 - accuracy: 0.7120
Epoch 62: val_loss did not improve from 0.92941
504/504 [==============================] - 43s 85ms/step - loss: 0.7871 - accuracy: 0.7120 - val_loss: 0.9769 - val_accuracy: 0.6444
Epoch 63/100
504/504 [==============================] - ETA: 0s - loss: 0.7765 - accuracy: 0.7165
Epoch 63: val_loss did not improve from 0.92941
504/504 [==============================] - 45s 90ms/step - loss: 0.7765 - accuracy: 0.7165 - val_loss: 1.0449 - val_accuracy: 0.6433
Epoch 64/100
504/504 [==============================] - ETA: 0s - loss: 0.7802 - accuracy: 0.7162
Epoch 64: val_loss did not improve from 0.92941
504/504 [==============================] - 43s 86ms/step - loss: 0.7802 - accuracy: 0.7162 - val_loss: 0.9632 - val_accuracy: 0.6678
Epoch 65/100
504/504 [==============================] - ETA: 0s - loss: 0.7681 - accuracy: 0.7232
Epoch 65: val_loss did not improve from 0.92941
504/504 [==============================] - 44s 87ms/step - loss: 0.7681 - accuracy: 0.7232 - val_loss: 0.9894 - val_accuracy: 0.6628
Epoch 66/100
504/504 [==============================] - ETA: 0s - loss: 0.7745 - accuracy: 0.7160
Epoch 66: val_loss did not improve from 0.92941
504/504 [==============================] - 43s 85ms/step - loss: 0.7745 - accuracy: 0.7160 - val_loss: 0.9723 - val_accuracy: 0.6550
Epoch 67/100
504/504 [==============================] - ETA: 0s - loss: 0.7700 - accuracy: 0.7193
Epoch 67: val_loss did not improve from 0.92941
504/504 [==============================] - 43s 85ms/step - loss: 0.7700 - accuracy: 0.7193 - val_loss: 1.0071 - val_accuracy: 0.6483
Epoch 68/100
504/504 [==============================] - ETA: 0s - loss: 0.7612 - accuracy: 0.7220
```

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

```
57/57 [==============================] - 1s 20ms/step - loss: 0.9230 - accuracy: 0.6747
Test Accuracy: 0.6746518015861511
```

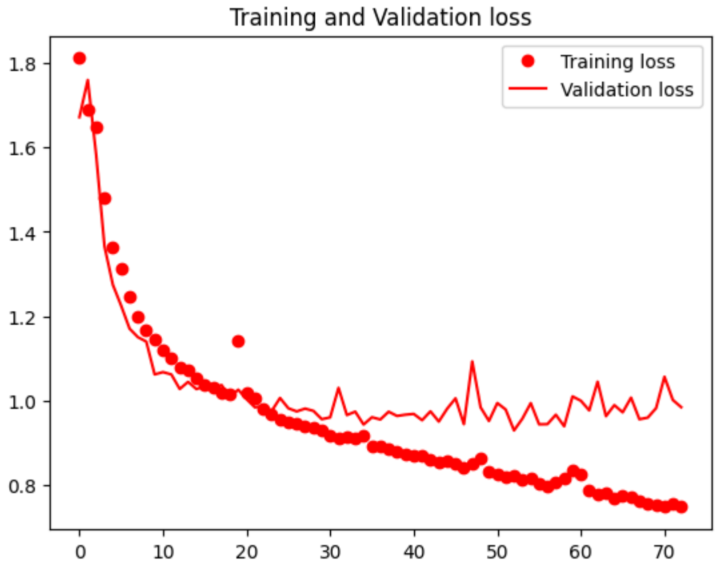
```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']


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 0x78ff60125510>
```



```
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 0x78ff6c0b6ce0>
```

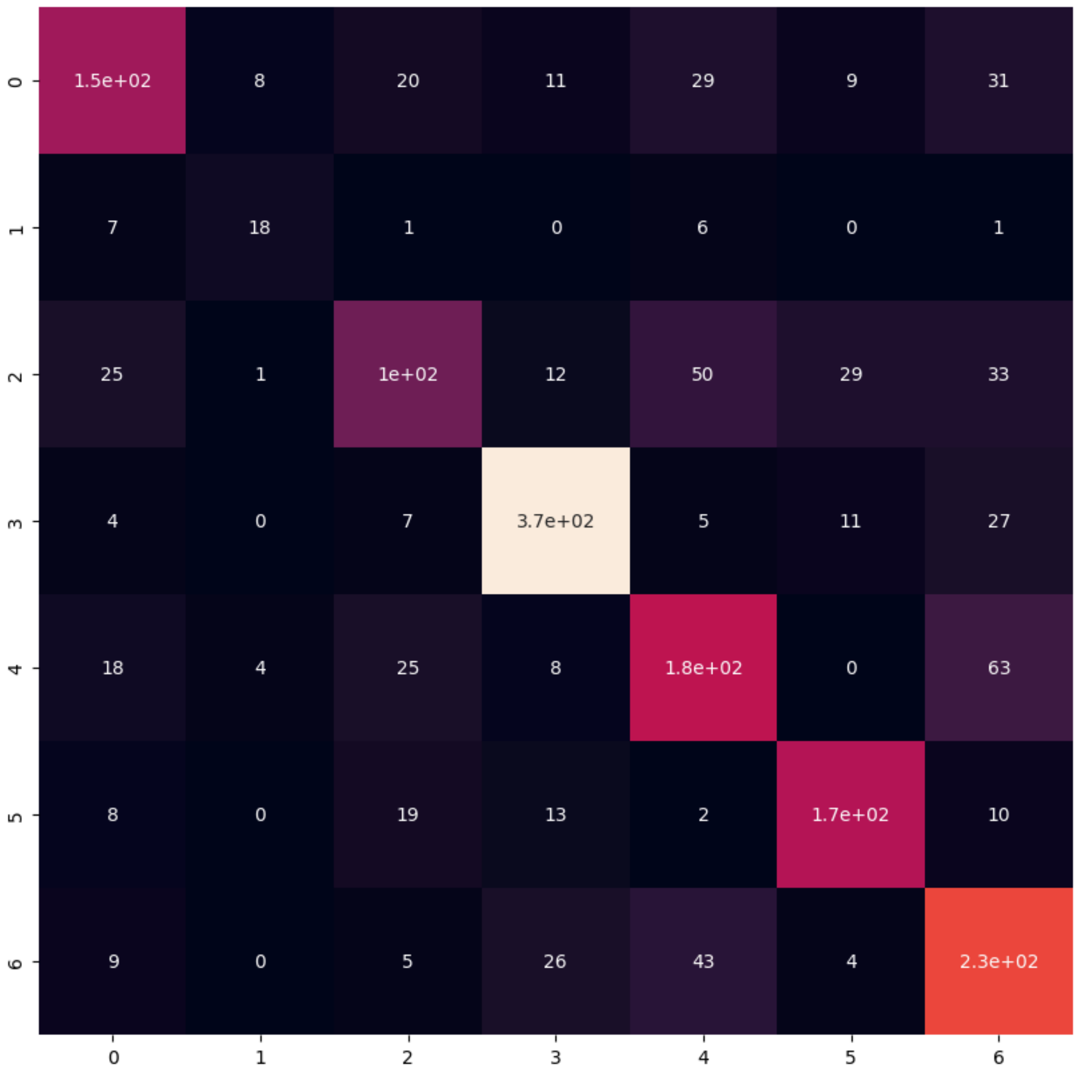


```
predicted_classes = np.argmax(VGGmodel.predict(X_test), axis=-1)
y_true = np.argmax(y_test, axis=-1)
```

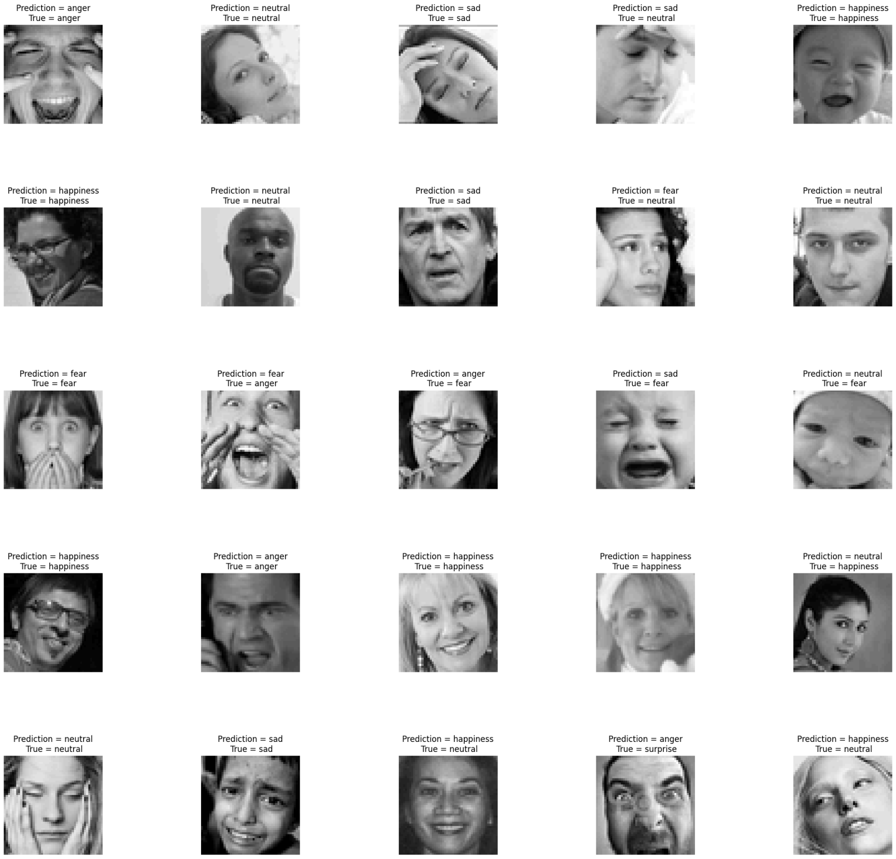
```
57/57 [==============================] - 1s 11ms/step
```

```
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], 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)
```



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