Importing Libraries

In [225]:
```
# To interact with operating system
import os

# For numerical computing
import numpy as np

# For Data Manipulation
import pandas as pd

# For visualization
import matplotlib.pyplot as plt
import seaborn as sns

# For computer vision tasks
import cv2 as cv

# Splitting data into training and testing set
from sklearn.model_selection import train_test_split

# Converts categorical column to numerical
from sklearn.preprocessing import LabelEncoder

# Converts class vectors(integrers) to binary classa
from keras.utils import to_categorical

# Applies transformations like rescaling and rotation to increase training
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# For Deep learning
import tensorflow as tf
from keras.models import Sequential, load_model
from keras.layers import Dense, Conv2D, Dropout, BatchNormalization, MaxPoo

# Optimizer for updating weigths
from keras.optimizers import Adam

# Fine-Tuning Model
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlate

# Model Evaluation Metrics
from sklearn.metrics import accuracy_score,classification_report
```

Loading Data

In [206]:
```
def create_df(data_dir,train=True):
    '''Creates a pandas dataframe with image path and labels

    Args:
    data_dir(str): path of the root directory
    train(bool): True if creating dataframe for train data.
                False if creating dataframe for validation data.

    Returns:
    pandas.DataFrame with 2 columns(image_path,labels)'''

    if train:
        data_dir = os.path.join(data_dir,'train')
    else:
        data_dir = os.path.join(data_dir,'validation')

    sub_folders = os.listdir(data_dir)
    print('Different classes are: {}'.format(sub_folders))

    images = []
    labels = []
    for sub_folder in sub_folders:
        label = sub_folder
        path = os.path.join(data_dir,sub_folder)
        image_dir = os.listdir(path)
        for image in image_dir:
            image_path = os.path.join(path,image)
            images.append(image_path)
            labels.append(label)
    dict = {'file_path':images,'label':labels}
    df = pd.DataFrame(dict)
    return df
```

In [207]:
```
root_dir = "data"
train_df = create_df(root_dir)
print("Training Size:",len(train_df))
print('Training Shape:',train_df.shape)
train_df.head()
```

```
Different classes are: ['angry', 'disgust', 'fear', 'happy', 'neutral', 's
ad', 'surprise']
Training Size: 28821
Training Shape: (28821, 2)
```

Out[207]:

	file_path	label
0	data\train\angry\0.jpg	angry
1	data\train\angry\1.jpg	angry
2	data\train\angry\10.jpg	angry
3	data\train\angry\10002.jpg	angry
4	data\train\angry\10016.jpg	angry

In [232]:
```
test_df = create_df(root_dir,train=False)
print("Testing Size:",len(test_df))
print('Testing Shape:',test_df.shape)
train_df.head()
```

```
Different classes are: ['angry', 'disgust', 'fear', 'happy', 'neutral', 's
ad', 'surprise']
Testing Size: 7066
Testing Shape: (7066, 2)
```

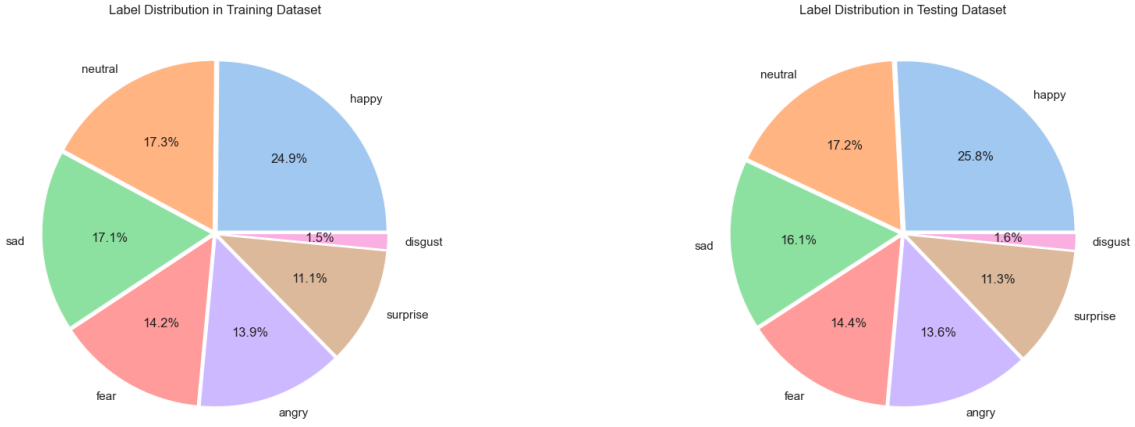
Out[232]:

	file_path	label
0	data\train\angry\0.jpg	angry
1	data\train\angry\1.jpg	angry
2	data\train\angry\10.jpg	angry
3	data\train\angry\10002.jpg	angry
4	data\train\angry\10016.jpg	angry

Visualization

Distribution of Labels

In [210]:
```
fig,axes = plt.subplots(nrows=1, ncols=2, figsize=(20,6))
sns.set_theme(style='darkgrid', palette='pastel')
color = sns.color_palette(palette='pastel')
explode = [0.02]*len(label_count)

# Training Data
label_count = train_df['label'].value_counts()
axes[0].pie(label_count.values,labels=label_count.index,autopct='%1.1f%%',
axes[0].set_title('Label Distribution in Training Dataset')

# Testing Data
label_count = test_df['label'].value_counts()
axes[1].pie(label_count.values,labels=label_count.index,autopct='%1.1f%%',
axes[1].set_title('Label Distribution in Testing Dataset')

plt.tight_layout()
plt.show()
```



Showing images

In [211]:
```
random_index = np.random.randint(0,len(train_df),16)
fig,axes = plt.subplots(nrows=4,ncols=4,figsize=(8,5),subplot_kw={'xticks':

for i,ax in enumerate(axes.flat):
  img = cv.imread(train_df['file_path'].iloc[random_index[i]])
  ax.imshow(img,cmap='Grays')
  ax.set_title(train_df['label'].iloc[random_index[i]])
plt.tight_layout()
plt.show()
```

Preprocessing

Splitting data

In [214]:
```
train,validate = train_test_split(train_df,test_size=0.2,random_state=42)
print('Train Shape:',train.shape)
print('Validate Shape:',validate.shape)
print('Test Shape:',test_df.shape)
```

```
Train Shape: (23056, 2)
Validate Shape: (5765, 2)
Test Shape: (7066, 2)
```

Data Augmentation

In [257]:
```
# For train
data_gen_train = ImageDataGenerator(rescale=1./255., width_shift_range=0.15
                                    shear_range=0.1, rotation_range=10, horizonta
train_set = data_gen_train.flow_from_dataframe(dataframe = train,x_col='fil
                                               seed=42,class_mode='categorical',col

# For validate and test
data_gen_test = ImageDataGenerator(rescale=1./255.)
validate_set = data_gen_test.flow_from_dataframe(dataframe = validate,x_col
                                               seed=42,class_mode='categorical',col
test_set = data_gen_test.flow_from_dataframe(dataframe = test_df,x_col='fil
                                               seed=42,class_mode='categorical',col
```

```
Found 23056 validated image filenames belonging to 7 classes.
Found 5765 validated image filenames belonging to 7 classes.
Found 7066 validated image filenames belonging to 7 classes.
```

In [218]:
```
# Building Model
def build_model():
    model = Sequential()
    # 1st Layer
    model.add(Conv2D(64, (5, 5), strides=(1, 1), padding='same', activation
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.3))

    # 2nd Layer
    model.add(Conv2D(128, (3, 3), strides=(1, 1), padding='same', activatio
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.3))

    # 3rd Layer
    model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activatio
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.3))

    # 4th Layer
    model.add(Conv2D(512, (3, 3), strides=(1, 1), padding='same', activatio
    model.add(BatchNormalization())
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.3))

    # Flatten Layer
    model.add(Flatten())

    # Fully connected layer 1
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))

    # Fully connected layer 2
    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.3))

    # Output Layer
    model.add(Dense(7, activation='softmax'))

    # Compiling the model
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_cr

    return model
```

In [219]:
```
model = build_model()
print(model.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_34 (Conv2D)          (None, 64, 64, 64)        1664

 batch_normalization_12 (Bat  (None, 64, 64, 64)       256
 chNormalization)

 max_pooling2d_20 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 32, 32, 64)        0

 conv2d_35 (Conv2D)          (None, 32, 32, 128)       73856

 batch_normalization_13 (Bat  (None, 32, 32, 128)      512
 chNormalization)

 max_pooling2d_21 (MaxPoolin  (None, 16, 16, 128)      0
 g2D)

 dropout_19 (Dropout)        (None, 16, 16, 128)       0

 conv2d_36 (Conv2D)          (None, 16, 16, 512)       590336

 batch_normalization_14 (Bat  (None, 16, 16, 512)      2048
 chNormalization)

 max_pooling2d_22 (MaxPoolin  (None, 8, 8, 512)        0
 g2D)

 dropout_20 (Dropout)        (None, 8, 8, 512)         0

 conv2d_37 (Conv2D)          (None, 8, 8, 512)         2359808

 batch_normalization_15 (Bat  (None, 8, 8, 512)        2048
 chNormalization)

 max_pooling2d_23 (MaxPoolin  (None, 4, 4, 512)        0
 g2D)

 dropout_21 (Dropout)        (None, 4, 4, 512)         0

 flatten_8 (Flatten)         (None, 8192)              0

 dense_18 (Dense)            (None, 256)               2097408

 batch_normalization_16 (Bat  (None, 256)              1024
 chNormalization)

 dropout_22 (Dropout)        (None, 256)               0

 dense_19 (Dense)            (None, 512)               131584

 batch_normalization_17 (Bat  (None, 512)              2048
 chNormalization)

 dropout_23 (Dropout)        (None, 512)               0

 dense_20 (Dense)            (None, 7)                 3591
```

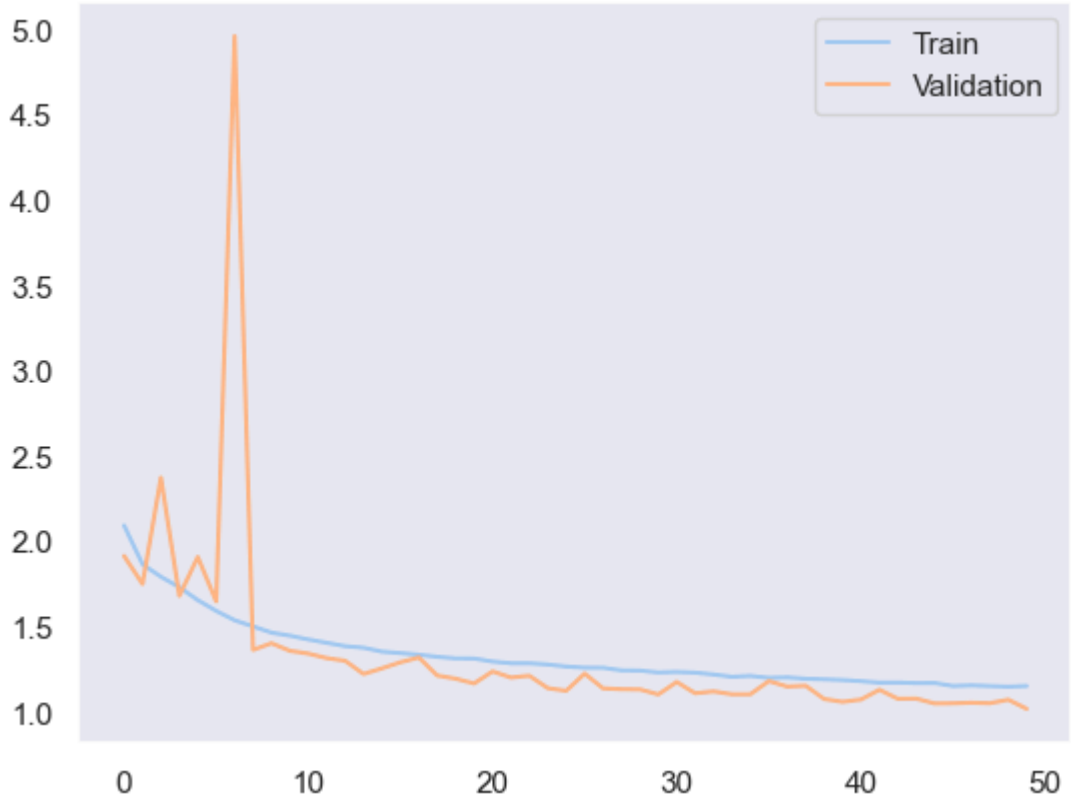
```
=================================================================
Total params: 5,266,183
Trainable params: 5,262,215
Non-trainable params: 3,968
_________________________________________________________________
None
```

In [107]:
```
%%time
history = model.fit(train_set,steps_per_epoch=len(train_set),validation_dat
                    epochs=50,verbose=1)
```

```
Epoch 1/50
721/721 [==============================] - 499s 691ms/step - loss: 2.09
06 - accuracy: 0.2094 - val_loss: 1.9121 - val_accuracy: 0.2342
Epoch 2/50
721/721 [==============================] - 455s 631ms/step - loss: 1.86
37 - accuracy: 0.2360 - val_loss: 1.7479 - val_accuracy: 0.2767
Epoch 3/50
721/721 [==============================] - 383s 531ms/step - loss: 1.78
99 - accuracy: 0.2761 - val_loss: 2.3712 - val_accuracy: 0.2611
Epoch 4/50
721/721 [==============================] - 357s 495ms/step - loss: 1.72
90 - accuracy: 0.3086 - val_loss: 1.6786 - val_accuracy: 0.3273
Epoch 5/50
721/721 [==============================] - 397s 551ms/step - loss: 1.65
33 - accuracy: 0.3463 - val_loss: 1.9063 - val_accuracy: 0.2843
Epoch 6/50
721/721 [==============================] - 426s 590ms/step - loss: 1.59
01 - accuracy: 0.3767 - val_loss: 1.6458 - val_accuracy: 0.3435
Epoch 7/50
```

In [108]:
```
model.save('model_1.h5')
```

In [220]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train','Validation'])
plt.grid()
plt.show()
```



In [266]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Train','Validation'])
plt.grid()
plt.show()
```

In [226]:
```
load_model = load_model('model_1.h5')
```

In [258]:
```
score = load_model.evaluate(test_set,steps=len(test_set),verbose=1)
print('Loss: {:.2f}'.format(score[0]))
print('Accuracy: {:.2f}'.format(score[1]))
```

```
221/221 [==============================] - 20s 89ms/step - loss: 1.0102 -
accuracy: 0.6248
Loss: 1.01
Accuracy: 0.62
```

In [259]:
```
pred = load_model.predict(test_set,verbose=0)
pred = np.argmax(pred,axis=1)

labels = (test_set.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]
```

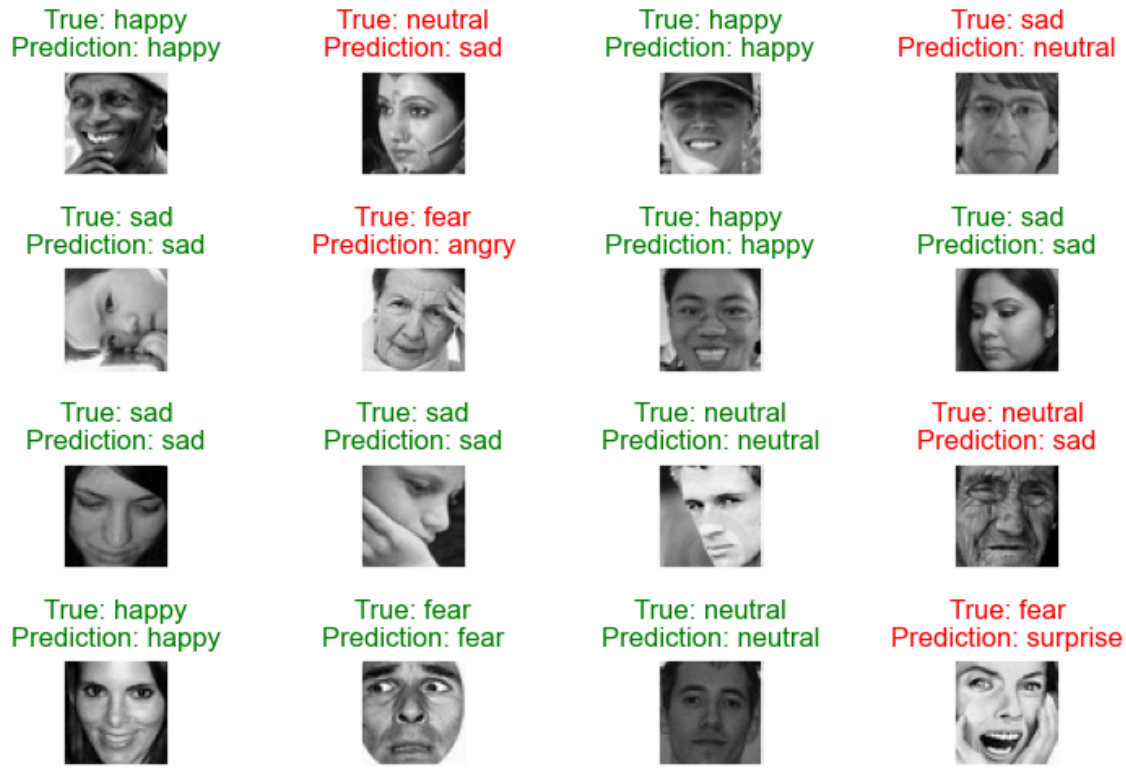
In [267]:
```
labels
```

Out[267]:
```
{0: 'angry',
 1: 'disgust',
 2: 'fear',
 3: 'happy',
 4: 'neutral',
 5: 'sad',
 6: 'surprise'}
```

In [260]:
```
y_test = list(test_df.label)
print('Accuracy Score: {:.3f}'.format(accuracy_score(y_test,pred)))
print(classification_report(y_test,pred))
```

```
Accuracy Score: 0.625
              precision    recall  f1-score   support

       angry       0.57      0.47      0.52       960
     disgust       0.65      0.38      0.48       111
        fear       0.54      0.32      0.40      1018
       happy       0.81      0.88      0.84      1825
     neutral       0.52      0.65      0.58      1216
         sad       0.47      0.56      0.51      1139
    surprise       0.74      0.72      0.73       797

    accuracy                           0.62      7066
   macro avg       0.61      0.57      0.58      7066
weighted avg       0.62      0.62      0.62      7066
```

In [265]:
```
index_random = np.random.randint(0,len(test_df),16)
fig,axes = plt.subplots(nrows=4, ncols=4,figsize=(8,5),subplot_kw={'xticks'

for i ,ax in enumerate(axes.flat):
  img = cv.imread(test_df['file_path'].iloc[index_random[i]])
  ax.imshow(img,cmap='Grays')
  if test_df['label'].iloc[index_random[i]] == pred[index_random[i]]:
    color = 'green'
  else:
    color = 'red'
  ax.set_title(f'True: {test_df.label.iloc[index_random[i]]}\nPrediction: {

plt.tight_layout()
plt.show()
```



In [ ]:
```
```