Imports

In this section, we import all the necessary libraries required for our project. These libraries provide the tools needed to manipulate data, perform mathematical calculations, visualize data, manage files, and build machine learning models.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')

import os
import tensorflow as tf
import keras
import cv2

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator, 
load_img, img_to_array
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, 
ReduceLROnPlateau
from tensorflow.keras.utils import plot_model
from tensorflow.keras import layers , models, optimizers

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.applications import ResNet50V2
```

Visualizing Classes

In this section of the code, we focus on organizing and visualizing the distribution of our dataset, specifically how many images we have for each emotional category in both training and testing datasets. This helps in understanding the balance of our data and preparing for effective model training. The output tables displayed provide a detailed count of images available for each emotion in both the training and testing datasets. Such visualization is essential for ensuring that our model trains on a balanced dataset, which is critical for maintaining accuracy across all emotional categories.

```
import os
import pandas as pd

# Use absolute paths
base_dir = 
os.path.expanduser('~/Downloads/EmotionBasedMusicRecommendationSystem/
```

```
EmotionBasedMusicRecommendationSystem/dataset')
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')

def Classes_Count(path, name):
    Classes_Dict = {}

    for Class in os.listdir(path):
        Full_Path = os.path.join(path, Class)
        Classes_Dict[Class] = len(os.listdir(Full_Path))

    df = pd.DataFrame(Classes_Dict, index=[name])

    return df

Train_Count = Classes_Count(train_dir,
'Train').transpose().sort_values(by="Train", ascending=False)
Test_Count = Classes_Count(test_dir,
'Test').transpose().sort_values(by="Test", ascending=False)

print("Train Count:\n", Train_Count)
print("Test Count:\n", Test_Count)
```

```
Train Count:
           Train
happy      7215
neutral    4965
sad        4830
fear       4097
angry      3995
surprise   3171
disgust     436
Test Count:
           Test
happy     1774
sad       1248
neutral   1233
fear      1024
angry      958
surprise   831
disgust    111
```

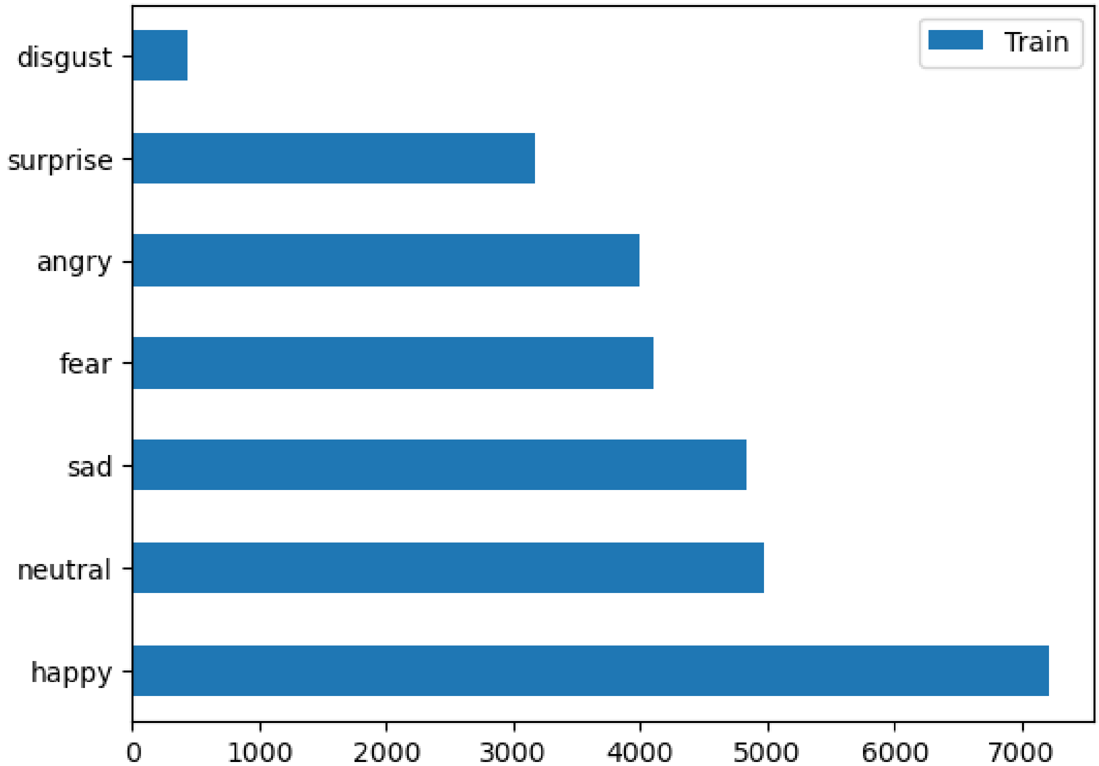
```
pd.concat([Train_Count,Test_Count] , axis=1)
```

	Train	Test
happy	7215	1774
neutral	4965	1233
sad	4830	1248
fear	4097	1024
angry	3995	958

```
surprise   3171   831
disgust     436   111
```

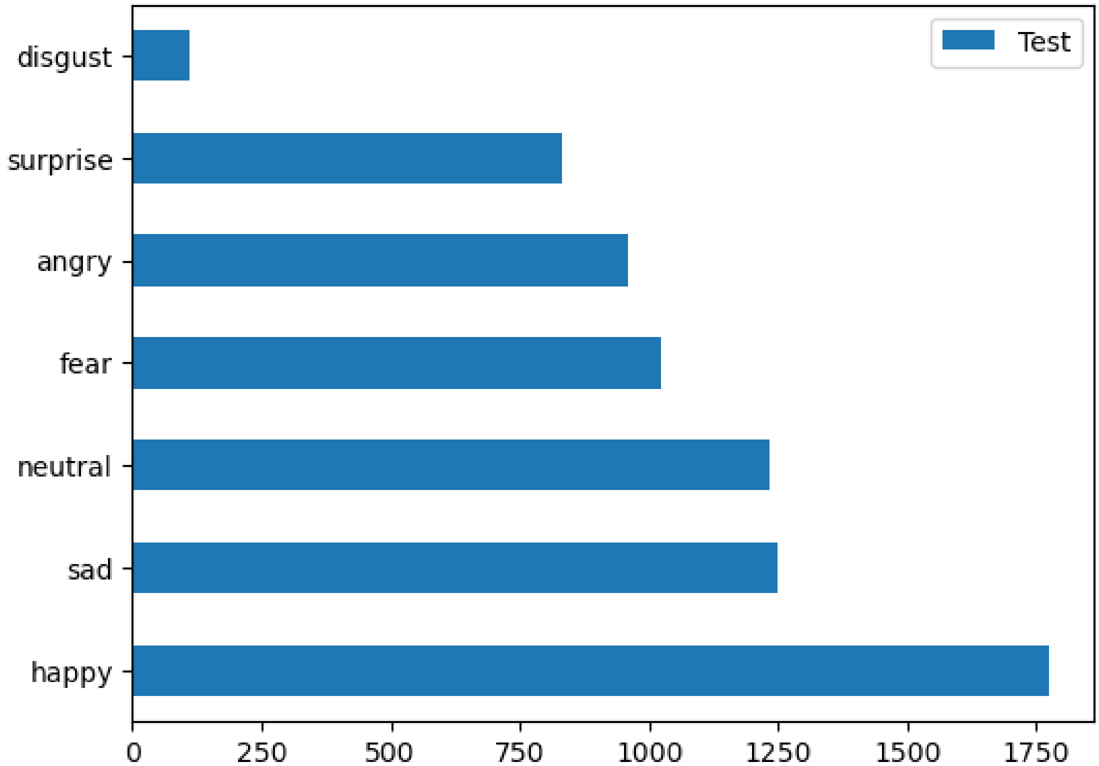
```
Train_Count.plot(kind='barh')
```

```
<Axes: >
```




```
Test_Count.plot(kind='barh')
```

```
<Axes: >
```




```
plt.style.use('default')
plt.figure(figsize = (25, 8))
image_count = 1
# Use the correct path
BASE_URL =
os.path.expanduser('~/Downloads/EmotionBasedMusicRecommendationSystem/
EmotionBasedMusicRecommendationSystem/dataset/train/')


for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(1, 7, image_count)
                image_count += 1
                image = cv2.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory, fontsize = 20)
```

This section of the code demonstrates how to visualize sample images from each class within our dataset. This is crucial for ensuring that the dataset is being read correctly and to get a visual sense of the different categories of emotions we are working with.

Data Preprocessing

This section of the code sets up the data preprocessing needed for training our machine learning model. It involves setting up the image data generators for both training and testing datasets, which include resizing, normalization, and data augmentation steps. The output from this script will confirm how many images are loaded and processed from each directory, showing a breakdown by class.

- This setup is crucial for training a well-performing model as it ensures that data is not only well-prepped but also augmented to enhance the model's ability to generalize from training data to real-world scenarios.

```
img_shape = 48
batch_size = 64
base_dir =
os.path.expanduser('~/Downloads/EmotionBasedMusicRecommendationSystem/
EmotionBasedMusicRecommendationSystem/dataset')
train_data_path = os.path.join(base_dir, 'train')
test_data_path = os.path.join(base_dir, 'test')

train_preprocessor = ImageDataGenerator(
        rescale = 1 / 255.,
        # Data Augmentation
        rotation_range=10,
        zoom_range=0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
    )


test_preprocessor = ImageDataGenerator(
    rescale = 1 / 255.,
)

train_data = train_preprocessor.flow_from_directory(
    train_data_path,
    class_mode="categorical",
```

```
    target_size=(img_shape,img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training',
)


test_data = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode="rgb",
    shuffle=False,
    batch_size=batch_size,
)
```

```
Found 28709 images belonging to 7 classes.
Found 7179 images belonging to 7 classes.
```

Building CNN Model

This section of the code defines the architecture of our convolutional neural network (CNN) used for emotion detection. The model is structured into multiple layers, each serving a specific function to process image data and classify it into categories corresponding to different emotions.

- **Sequential Model**: which allows us to build a model layer by layer

Dense and Output Layers

- After flattening the output from the convolutional and pooling layers, the data is passed through fully connected layers (`Dense`).
- **Dense Layers**:each node in the dense layer receives input from all nodes of its preceding layer. They are used to classify the features extracted by the convolutions and pooling into the final output categories based on the learned weights.
- **Activation Functions**: 'relu' is used for non-linear transformations within the dense layers, helping the network learn complex patterns in the data. The final layer uses 'softmax' activation function to output probabilities of the classes which sum to one.

Summary

- The `model.summary()` method is called to display the architecture of the model including the types of layers used, their parameters, and output shapes. This summary is crucial for understanding the complexity of the model and debugging the layer dimensions.

```python
def Create_CNN_Model():

    model = Sequential()

    #CNN1
    model.add(Conv2D(32, (3,3), activation='relu',
input_shape=(img_shape, img_shape, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.25))

    #CNN2
    model.add(Conv2D(64, (3,3), activation='relu', ))
    model.add(BatchNormalization())
    model.add(Conv2D(128,(3,3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.25))

    #CNN3
    model.add(Conv2D(128, (3,3), activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(256,(3,3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.25))


    #Output
    model.add(Flatten())

    model.add(Dense(1024, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(64, activation='relu'))
```

```
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

    model.add(Dense(32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.25))

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


    return model

CNN_Model = Create_CNN_Model()

CNN_Model.summary()

CNN_Model.compile(optimizer="adam", loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
C:\Users\Manahil\Anaconda3\lib\site-packages\keras\src\layers\
convolutional\base_conv.py:99: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape               ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━━━━┩
│ conv2d (Conv2D)                  │ (None, 46, 46, 32)         │
896 │
├──────────────────────────────────┼────────────────────────────┼───────
────────┤
│ batch_normalization              │ (None, 46, 46, 32)         │
128 │
│ (BatchNormalization)             │                            │
│
├──────────────────────────────────┼────────────────────────────┼───────
────────┤
│ conv2d_1 (Conv2D)                │ (None, 46, 46, 64)         │
18,496 │
├──────────────────────────────────┼────────────────────────────┼───────
────────┤
│ batch_normalization_1            │ (None, 46, 46, 64)         │
256 │
```

```
│ (BatchNormalization)              │                           │
│
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ max_pooling2d (MaxPooling2D)      │ (None, 23, 23, 64)        │
0 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ dropout (Dropout)                 │ (None, 23, 23, 64)        │
0 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ conv2d_2 (Conv2D)                 │ (None, 21, 21, 64)        │
36,928 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ batch_normalization_2             │ (None, 21, 21, 64)        │
256 │
│ (BatchNormalization)              │                           │
│
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ conv2d_3 (Conv2D)                 │ (None, 21, 21, 128)       │
73,856 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ batch_normalization_3             │ (None, 21, 21, 128)       │
512 │
│ (BatchNormalization)              │                           │
│
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ max_pooling2d_1 (MaxPooling2D)    │ (None, 11, 11, 128)       │
0 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ dropout_1 (Dropout)               │ (None, 11, 11, 128)       │
0 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ conv2d_4 (Conv2D)                 │ (None, 9, 9, 128)         │
147,584 │
├───────────────────────────────────┼───────────────────────────┼───────
───────┤
│ batch_normalization_4             │ (None, 9, 9, 128)         │
512 │
│ (BatchNormalization)              │                           │
│
├───────────────────────────────────┼───────────────────────────┼───────
```

```
────────┤
│ conv2d_5 (Conv2D)                   │ (None, 9, 9, 256)           │
295,168 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ batch_normalization_5               │ (None, 9, 9, 256)           │
1,024 │
│ (BatchNormalization)                │                             │
│
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ max_pooling2d_2 (MaxPooling2D)      │ (None, 5, 5, 256)           │
0 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ dropout_2 (Dropout)                 │ (None, 5, 5, 256)           │
0 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ flatten (Flatten)                   │ (None, 6400)                │
0 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ dense (Dense)                       │ (None, 1024)                │
6,554,624 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ batch_normalization_6               │ (None, 1024)                │
4,096 │
│ (BatchNormalization)                │                             │
│
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ dropout_3 (Dropout)                 │ (None, 1024)                │
0 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ dense_1 (Dense)                     │ (None, 512)                 │
524,800 │
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ batch_normalization_7               │ (None, 512)                 │
2,048 │
│ (BatchNormalization)                │                             │
│
├─────────────────────────────────────┼─────────────────────────────┼────────
────────┤
│ dropout_4 (Dropout)                 │ (None, 512)                 │
0 │
```

```
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dense_2 (Dense)                 │ (None, 256)                │
131,328 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ batch_normalization_8           │ (None, 256)                │
1,024 │
│ (BatchNormalization)            │                            │
│
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dropout_5 (Dropout)             │ (None, 256)                │
0 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dense_3 (Dense)                 │ (None, 128)                │
32,896 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ batch_normalization_9           │ (None, 128)                │
512 │
│ (BatchNormalization)            │                            │
│
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dropout_6 (Dropout)             │ (None, 128)                │
0 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dense_4 (Dense)                 │ (None, 64)                 │
8,256 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ batch_normalization_10          │ (None, 64)                 │
256 │
│ (BatchNormalization)            │                            │
│
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dropout_7 (Dropout)             │ (None, 64)                 │
0 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ dense_5 (Dense)                 │ (None, 32)                 │
2,080 │
├─────────────────────────────────┼────────────────────────────┼───────
───────┤
│ batch_normalization_11          │ (None, 32)                 │
```

```
128 │
│ (BatchNormalization)                  │                            │
│
├───────────────────────────────────────┼────────────────────────────┼───────
──────┤
│ dropout_8 (Dropout)                   │ (None, 32)                 │
0 │
├───────────────────────────────────────┼────────────────────────────┼───────
──────┤
│ dense_6 (Dense)                       │ (None, 7)                  │
231 │
└───────────────────────────────────────┴────────────────────────────┴───────
──────┘

 Total params: 7,837,895 (29.90 MB)

 Trainable params: 7,832,519 (29.88 MB)

 Non-trainable params: 5,376 (21.00 KB)
```

Specifying Callbacks

```
# Create Callback Checkpoint with .keras extension
checkpoint_path = "CNN_Model_Checkpoint.keras"

Checkpoint = ModelCheckpoint(checkpoint_path, monitor="val_accuracy",
save_best_only=True)

# Create Early Stopping Callback to monitor the accuracy
Early_Stopping = EarlyStopping(monitor='val_accuracy', patience=15,
restore_best_weights=True, verbose=1)
# Create ReduceLROnPlateau Callback to reduce overfitting by
decreasing learning rate
Reducing_LR =
tf.keras.callbacks.ReduceLROnPlateau( monitor='val_loss',
                                                  factor=0.2,
                                                  patience=2,
#                                                   min_lr=0.000005,
                                                  verbose=1)

callbacks = [Early_Stopping, Reducing_LR]

steps_per_epoch = train_data.n // train_data.batch_size
validation_steps = test_data.n // test_data.batch_size

# Create Callback Checkpoint with .keras extension
checkpoint_path = "CNN_Model_Checkpoint.keras"

Checkpoint = ModelCheckpoint(checkpoint_path, monitor="val_accuracy",
save_best_only=True)
```

```
# Create Early Stopping Callback with increased patience
Early_Stopping = EarlyStopping(monitor='val_accuracy', patience=10,
restore_best_weights=True, verbose=1)

# Create ReduceLROnPlateau Callback with adjusted parameters
Reducing_LR = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,  # Reduce learning rate by a smaller factor
    patience=5,  # Allow more epochs before reducing learning rate
    min_lr=1e-7,  # Set a minimum learning rate
    verbose=1
)

callbacks = [Checkpoint, Early_Stopping, Reducing_LR]

steps_per_epoch = train_data.n // train_data.batch_size
validation_steps = test_data.n // test_data.batch_size
```

Learning Rate Adjustments:

- `ReduceLROnPlateau`: This callback reduces the learning rate when a metric has stopped improving.
- **Plateaus and Adjustments**: The model occasionally hits plateaus where the accuracy does not improve. These are often followed by adjustments in the learning rate, as indicated by `ReduceLROnPlateau`, which helps in overcoming these plateaus.

```
CNN_history = CNN_Model.fit( train_data , validation_data= test_data ,
epochs=50, batch_size= batch_size,
                            callbacks=callbacks, steps_per_epoch=
steps_per_epoch, validation_steps=validation_steps)

Epoch 1/50

C:\Users\Manahil\Anaconda3\lib\site-packages\keras\src\trainers\
data_adapters\py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

448/448 ━━━━━━━━━━━━━━━━━━━━ 393s 853ms/step - accuracy: 0.1797 -
loss: 2.1952 - val_accuracy: 0.2577 - val_loss: 1.8346 -
learning_rate: 0.0010
Epoch 2/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 205us/step - accuracy: 0.2031 - loss:
0.9217 - val_accuracy: 0.0909 - val_loss: 1.2791 - learning_rate:
0.0010
Epoch 3/50
```

```
C:\Users\Manahil\Anaconda3\lib\contextlib.py:137: UserWarning: Your
input ran out of data; interrupting training. Make sure that your
dataset or generator can generate at least `steps_per_epoch * epochs`
batches. You may need to use the `.repeat()` function when building
your dataset.
  self.gen.throw(typ, value, traceback)

448/448 ━━━━━━━━━━━━━━━━━━━━ 166s 369ms/step - accuracy: 0.2744 -
loss: 1.7886 - val_accuracy: 0.3641 - val_loss: 1.6057 -
learning_rate: 0.0010
Epoch 4/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 1s 740us/step - accuracy: 0.2812 - loss:
0.8796 - val_accuracy: 0.8182 - val_loss: 0.4118 - learning_rate:
0.0010
Epoch 5/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 167s 372ms/step - accuracy: 0.3552 -
loss: 1.6338 - val_accuracy: 0.4068 - val_loss: 1.5267 -
learning_rate: 0.0010
Epoch 6/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 80us/step - accuracy: 0.3750 - loss:
0.7698 - val_accuracy: 0.7273 - val_loss: 0.5739 - learning_rate:
0.0010
Epoch 7/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 168s 374ms/step - accuracy: 0.4076 -
loss: 1.5063 - val_accuracy: 0.4400 - val_loss: 1.4772 -
learning_rate: 0.0010
Epoch 8/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 82us/step - accuracy: 0.5156 - loss:
0.6933 - val_accuracy: 0.8182 - val_loss: 0.4140 - learning_rate:
0.0010
Epoch 9/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 347ms/step - accuracy: 0.4493 - loss:
1.4222
Epoch 9: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.
448/448 ━━━━━━━━━━━━━━━━━━━━ 163s 362ms/step - accuracy: 0.4493 -
loss: 1.4222 - val_accuracy: 0.4965 - val_loss: 1.3089 -
learning_rate: 0.0010
Epoch 10/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 82us/step - accuracy: 0.4688 - loss:
0.6944 - val_accuracy: 0.8182 - val_loss: 0.4203 - learning_rate:
5.0000e-04
Epoch 11/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 164s 366ms/step - accuracy: 0.4868 -
loss: 1.3416 - val_accuracy: 0.5213 - val_loss: 1.2402 -
learning_rate: 5.0000e-04
Epoch 12/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 1s 770us/step - accuracy: 0.4375 - loss:
0.8402 - val_accuracy: 0.9091 - val_loss: 0.1709 - learning_rate:
5.0000e-04
```

```
Epoch 13/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 164s 365ms/step - accuracy: 0.5106 -
loss: 1.2978 - val_accuracy: 0.5405 - val_loss: 1.1884 -
learning_rate: 5.0000e-04
Epoch 14/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 91us/step - accuracy: 0.5469 - loss:
0.6389 - val_accuracy: 0.9091 - val_loss: 0.1090 - learning_rate:
5.0000e-04
Epoch 15/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 164s 365ms/step - accuracy: 0.5250 -
loss: 1.2628 - val_accuracy: 0.5605 - val_loss: 1.1582 -
learning_rate: 5.0000e-04
Epoch 16/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 1s 671us/step - accuracy: 0.6250 - loss:
0.5466 - val_accuracy: 1.0000 - val_loss: 0.1687 - learning_rate:
5.0000e-04
Epoch 17/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 162s 361ms/step - accuracy: 0.5364 -
loss: 1.2391 - val_accuracy: 0.5481 - val_loss: 1.1939 -
learning_rate: 5.0000e-04
Epoch 18/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 77us/step - accuracy: 0.5156 - loss:
0.6365 - val_accuracy: 1.0000 - val_loss: 0.1263 - learning_rate:
5.0000e-04
Epoch 19/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 345ms/step - accuracy: 0.5433 - loss:
1.2253
Epoch 19: ReduceLROnPlateau reducing learning rate to
0.0002500000118743628.
448/448 ━━━━━━━━━━━━━━━━━━━━ 162s 360ms/step - accuracy: 0.5433 -
loss: 1.2253 - val_accuracy: 0.5532 - val_loss: 1.1809 -
learning_rate: 5.0000e-04
Epoch 20/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 78us/step - accuracy: 0.5312 - loss:
0.6551 - val_accuracy: 0.9091 - val_loss: 0.1868 - learning_rate:
2.5000e-04
Epoch 21/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 162s 360ms/step - accuracy: 0.5551 -
loss: 1.1969 - val_accuracy: 0.5911 - val_loss: 1.0790 -
learning_rate: 2.5000e-04
Epoch 22/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 86us/step - accuracy: 0.5469 - loss:
0.6605 - val_accuracy: 1.0000 - val_loss: 0.0912 - learning_rate:
2.5000e-04
Epoch 23/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 161s 360ms/step - accuracy: 0.5716 -
loss: 1.1516 - val_accuracy: 0.6031 - val_loss: 1.0524 -
learning_rate: 2.5000e-04
Epoch 24/50
```

```
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 87us/step - accuracy: 0.6406 - loss:
0.5266 - val_accuracy: 1.0000 - val_loss: 0.0690 - learning_rate:
2.5000e-04
Epoch 25/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 162s 360ms/step - accuracy: 0.5820 -
loss: 1.1403 - val_accuracy: 0.6021 - val_loss: 1.0507 -
learning_rate: 2.5000e-04
Epoch 26/50
448/448 ━━━━━━━━━━━━━━━━━━━━ 0s 104us/step - accuracy: 0.5469 - loss:
0.5433 - val_accuracy: 1.0000 - val_loss: 0.0516 - learning_rate:
2.5000e-04
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 16.
```

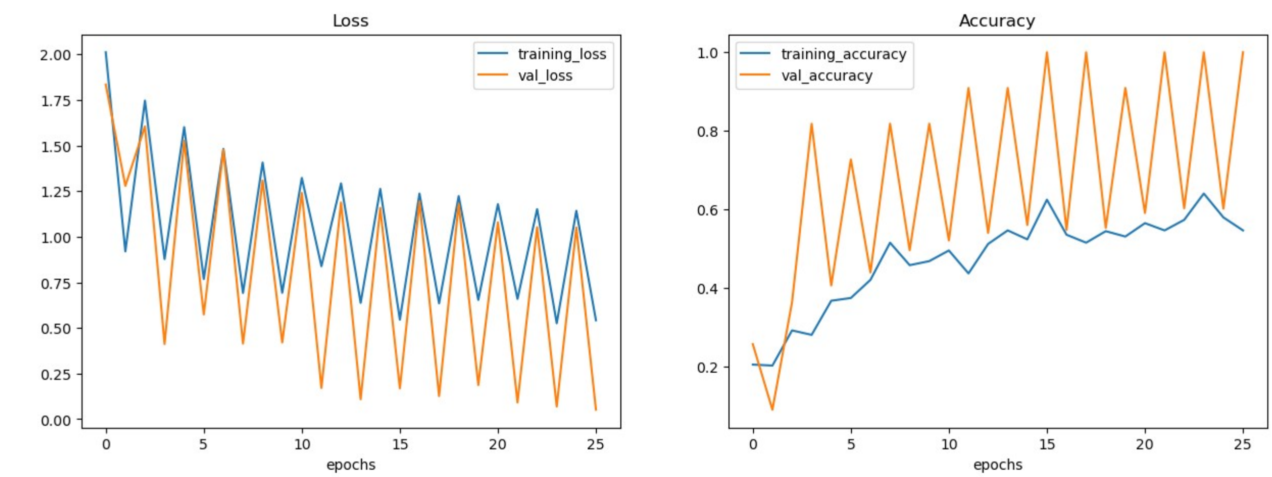
Evaluating CNN Model

```
CNN_Score = CNN_Model.evaluate(test_data)

print("    Test Loss: {:.5f}".format(CNN_Score[0]))
print("Test Accuracy: {:.2f}%".format(CNN_Score[1] * 100))
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 9s 83ms/step - accuracy: 0.4591 - loss:
1.3807
    Test Loss: 1.15648
Test Accuracy: 56.00%
```

```
def plot_curves(history):

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    accuracy = history.history["accuracy"]
    val_accuracy = history.history["val_accuracy"]

    epochs = range(len(history.history["loss"]))

    plt.figure(figsize=(15,5))

    #plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label = "training_loss")
    plt.plot(epochs, val_loss, label = "val_loss")
    plt.title("Loss")
    plt.xlabel("epochs")
    plt.legend()

    #plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label = "training_accuracy")
```

```
    plt.plot(epochs, val_accuracy, label = "val_accuracy")
    plt.title("Accuracy")
    plt.xlabel("epochs")
    plt.legend()

  #plt.tight_layout()

plot_curves(CNN_history)
```




```
CNN_Predictions = CNN_Model.predict(test_data)

# Choosing highest probalbilty class in every prediction
CNN_Predictions = np.argmax(CNN_Predictions, axis=1)
```

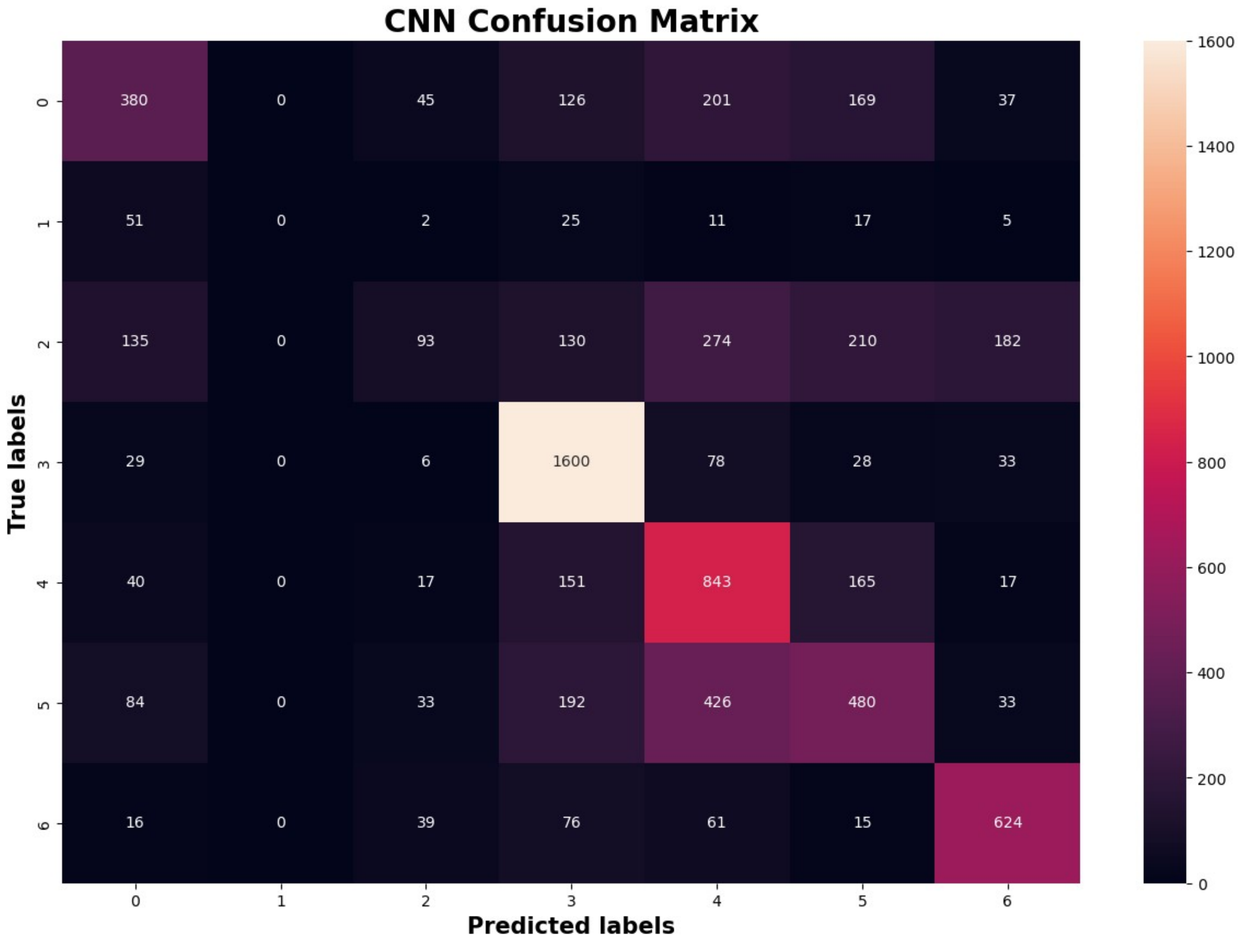
```
113/113 ━━━━━━━━━━━━━━━━━━━━ 8s 65ms/step
```

```
test_data.class_indices
```

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

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

fig, ax= plt.subplots(figsize=(15,10))

cm=confusion_matrix(test_data.labels, CNN_Predictions)

sns.heatmap(cm, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15, fontweight='bold')
```

```
ax.set_ylabel('True labels', fontsize=15, fontweight='bold')
ax.set_title('CNN Confusion Matrix', fontsize=20, fontweight='bold')

Text(0.5, 1.0, 'CNN Confusion Matrix')
```



ResNet50V2 Model

A ResNet50V2 model, which is a more complex and powerful model compared to traditional CNNs. The logs at the end indicate the number of images processed, confirming the effective loading and augmentation of the dataset. This output is essential to verify that the data pipeline is functioning as expected.

```
# specifing new image shape for resnet
img_shape = 224
batch_size = 64
base_dir =
os.path.expanduser('~/Downloads/EmotionBasedMusicRecommendationSystem/
EmotionBasedMusicRecommendationSystem/dataset')
```

```
train_data_path = os.path.join(base_dir, 'train')
test_data_path = os.path.join(base_dir, 'test')

train_preprocessor = ImageDataGenerator(
        rescale = 1 / 255.,
        rotation_range=10,
        zoom_range=0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
    )


test_preprocessor = ImageDataGenerator(
    rescale = 1 / 255.,
)

train_data = train_preprocessor.flow_from_directory(
    train_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training',
)

test_data = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode="rgb",
    shuffle=False,
    batch_size=batch_size,
)
```

```
Found 28709 images belonging to 7 classes.
Found 7179 images belonging to 7 classes.
```

Fine-Tuning ResNet50V2

Outlines the setup and training process for fine-tuning the ResNet50V2 model for emotion detection.

Training Execution

- The model is trained using the `fit` method, with specified `steps_per_epoch` and `validation_steps` from the training and validation data

Summary and Insights

- **Model Summary**: Outputs the structure and parameter count of the newly fine-tuned model.

```
ResNet50V2 = tf.keras.applications.ResNet50V2(input_shape=(224, 224, 3),
                                               include_top= False,
                                               weights='imagenet'
                                               )



ResNet50V2.trainable = True

for layer in ResNet50V2.layers[:-50]:
    layer.trainable = False

def Create_ResNet50V2_Model():

    model = Sequential([
                      ResNet50V2,
                      Dropout(.25),
                      BatchNormalization(),
                      Flatten(),
                      Dense(64, activation='relu'),
                      BatchNormalization(),
                      Dropout(.5),
                      Dense(7,activation='softmax')
                    ])
    return model

ResNet50V2_Model = Create_ResNet50V2_Model()

ResNet50V2_Model.summary()

ResNet50V2_Model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape               ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━
━━━━━┩
│ resnet50v2 (Functional)          │ ?                          │
23,564,800 │
├──────────────────────────────────┼────────────────────────────┼───────────
─────┤
│ dropout (Dropout)                │ ?                          │
```

```
0 │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ batch_normalization                │ ?                           │
0 │
│ (BatchNormalization)               │                             │
(unbuilt) │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ flatten (Flatten)                  │ ?                           │
0 │
│                                    │                             │
(unbuilt) │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ dense (Dense)                      │ ?                           │
0 │
│                                    │                             │
(unbuilt) │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ batch_normalization_1              │ ?                           │
0 │
│ (BatchNormalization)               │                             │
(unbuilt) │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ dropout_1 (Dropout)                │ ?                           │
0 │
├────────────────────────────────────┼─────────────────────────────┼─────────
───────┤
│ dense_1 (Dense)                    │ ?                           │
0 │
│                                    │                             │
(unbuilt) │
└────────────────────────────────────┴─────────────────────────────┴─────────
───────┘

 Total params: 23,564,800 (89.89 MB)

 Trainable params: 16,352,256 (62.38 MB)

 Non-trainable params: 7,212,544 (27.51 MB)
```

Specifying Callbacks

```
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

```python
# Create Callback Checkpoint with .keras extension
checkpoint_path = "ResNet50V2_Model_Checkpoint.keras"

Checkpoint = ModelCheckpoint(checkpoint_path, monitor="val_accuracy",
save_best_only=True)

# Create Early Stopping Callback to monitor the accuracy
Early_Stopping = EarlyStopping(monitor='val_accuracy', patience=7,
restore_best_weights=True, verbose=1)

# Create ReduceLROnPlateau Callback to reduce overfitting by
decreasing learning
Reducing_LR = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    verbose=1
)

callbacks = [Checkpoint, Early_Stopping, Reducing_LR]

steps_per_epoch = train_data.n // train_data.batch_size
validation_steps = test_data.n // test_data.batch_size

ResNet50V2_history = ResNet50V2_Model.fit(train_data ,validation_data
= test_data , epochs=30, batch_size=batch_size,
                                         callbacks = callbacks,
steps_per_epoch=steps_per_epoch, validation_steps=validation_steps)
```

```
Epoch 1/30

C:\Users\Manahil\Anaconda3\lib\site-packages\keras\src\trainers\
data_adapters\py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

448/448 ━━━━━━━━━━━━━━━━━━━━ 1565s 3s/step - accuracy: 0.4115 - loss:
1.7882 - val_accuracy: 0.5395 - val_loss: 2.8305 - learning_rate:
0.0010
Epoch 2/30
  1/448 ━━━━━━━━━━━━━━━━━━━━ 21:47 3s/step - accuracy: 0.6719 - loss:
1.0369

C:\Users\Manahil\Anaconda3\lib\contextlib.py:137: UserWarning: Your
input ran out of data; interrupting training. Make sure that your
dataset or generator can generate at least `steps_per_epoch * epochs`
batches. You may need to use the `.repeat()` function when building
```

```
your dataset.
  self.gen.throw(typ, value, traceback)

448/448 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.6719 - loss:
0.5196 - val_accuracy: 0.8182 - val_loss: 0.3391 - learning_rate:
0.0010
Epoch 3/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 1559s 3s/step - accuracy: 0.5550 - loss:
1.2213 - val_accuracy: 0.5935 - val_loss: 1.1233 - learning_rate:
0.0010
Epoch 4/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.6875 - loss:
0.6751 - val_accuracy: 1.0000 - val_loss: 0.1169 - learning_rate:
0.0010
Epoch 5/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 1532s 3s/step - accuracy: 0.5950 - loss:
1.1179 - val_accuracy: 0.5942 - val_loss: 1.0663 - learning_rate:
0.0010
Epoch 6/30
  1/448 ━━━━━━━━━━━━━━━━━━━━ 22:09 3s/step - accuracy: 0.5156 - loss:
1.3421
Epoch 6: ReduceLROnPlateau reducing learning rate to
0.0002000000000949949026.
448/448 ━━━━━━━━━━━━━━━━━━━━ 3s 879us/step - accuracy: 0.5156 - loss:
0.6725 - val_accuracy: 1.0000 - val_loss: 0.1944 - learning_rate:
0.0010
Epoch 7/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 1531s 3s/step - accuracy: 0.6269 - loss:
1.0232 - val_accuracy: 0.6530 - val_loss: 0.9482 - learning_rate:
2.0000e-04
Epoch 8/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 3s 858us/step - accuracy: 0.6562 - loss:
0.4615 - val_accuracy: 1.0000 - val_loss: 0.0977 - learning_rate:
2.0000e-04
Epoch 9/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 1529s 3s/step - accuracy: 0.6543 - loss:
0.9563 - val_accuracy: 0.6585 - val_loss: 0.9323 - learning_rate:
2.0000e-04
Epoch 10/30
  1/448 ━━━━━━━━━━━━━━━━━━━━ 21:22 3s/step - accuracy: 0.7344 - loss:
0.7173
Epoch 10: ReduceLROnPlateau reducing learning rate to
4.0000001899898055e-05.
448/448 ━━━━━━━━━━━━━━━━━━━━ 3s 864us/step - accuracy: 0.7344 - loss:
0.3595 - val_accuracy: 1.0000 - val_loss: 0.1371 - learning_rate:
2.0000e-04
Epoch 11/30
448/448 ━━━━━━━━━━━━━━━━━━━━ 1532s 3s/step - accuracy: 0.6727 - loss:
0.9141 - val_accuracy: 0.6641 - val_loss: 0.9075 - learning_rate:
4.0000e-05
```

```
Epoch 11: early stopping
Restoring model weights from the end of the best epoch: 4.
```

FineTuning Further

This section shows the process of fine-tuning and training the ResNet50V2 model with custom top layers for emotion detection. The last run took over 6 hours and the accuracy was not up to standard. A series of custom layers are added on top of the base ResNet50V2 model to tailor it for emotion classification. The model is trained using the `fit` function with specified `epochs` and `callbacks`, closely monitoring the validation loss and accuracy. Each epoch's progress is logged, showing the accuracy, loss, validation accuracy, and validation loss, providing insights into the model's performance and the effectiveness of the learning rate adjustments.

```
# Data augmentation and preprocessing setup
train_preprocessor = ImageDataGenerator(
    rescale=1/255.,
    rotation_range=40,
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.1  # Using a subset for validation
)

test_preprocessor = ImageDataGenerator(
    rescale=1/255.
)

# Load a subset of the data for training and testing
train_data = train_preprocessor.flow_from_directory(
    train_data_path,
    class_mode="categorical",
    target_size=(img_shape, img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training'
)

test_data = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape, img_shape),
    color_mode="rgb",
```

```
    shuffle=False,
    batch_size=batch_size
)
```

```
Found 25841 images belonging to 7 classes.
Found 7179 images belonging to 7 classes.
```

```
# Loading and configuring the ResNet50V2 model
ResNet50V2 = tf.keras.applications.ResNet50V2(
    input_shape=(img_shape, img_shape, 3),
    include_top=False,
    weights='imagenet'
)

# Freezing layers for faster training
ResNet50V2.trainable = True
for layer in ResNet50V2.layers[:-50]:
    layer.trainable = False

def Create_ResNet50V2_Model():
    model = Sequential([
        ResNet50V2,
        Flatten(),
        BatchNormalization(),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(7, activation='softmax')
    ])
    return model

ResNet50V2_Model = Create_ResNet50V2_Model()

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.regularizers import l2

def Create_ResNet50V2_Model():
    base_model = ResNet50V2(
        include_top=False,
        weights='imagenet',
        input_shape=(128, 128, 3)  # Adjusted to match the resized
images
    )
    base_model.trainable = True

    # Freeze layers as previously described
```

```
    for layer in base_model.layers[:-50]:
        layer.trainable = False

    model = Sequential([
        base_model,
        Flatten(),
        BatchNormalization(),
        Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),
        Dense(7, activation='softmax')
    ])
    return model

model = Create_ResNet50V2_Model()

initial_learning_rate = 0.001
optimizer =
tf.keras.optimizers.Adam(learning_rate=initial_learning_rate)

model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

def custom_lr_scheduler(epoch, lr):
    if epoch < 10:
        return lr  # keep the initial learning rate for the first 10
epochs
    elif epoch < 20:
        return 0.0001  # Reduce to 0.0001 after 10 epochs
    else:
        return 0.00001  # Reduce further after 20 epochs

from tensorflow.keras.callbacks import LearningRateScheduler,
EarlyStopping

lr_scheduler = LearningRateScheduler(custom_lr_scheduler, verbose=1)
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5,
restore_best_weights=True)

callbacks = [lr_scheduler, early_stopping]



history = model.fit(train_data,
                    validation_data=test_data,
                    epochs=40,
                    callbacks=callbacks)


Epoch 1: LearningRateScheduler setting learning rate to
0.0010000000474974513.
```

```
Epoch 1/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 525s 1s/step - accuracy: 0.3553 - loss:
3.0098 - val_accuracy: 0.5289 - val_loss: 1.5353 - learning_rate:
0.0010

Epoch 2: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 2/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 507s 1s/step - accuracy: 0.5047 - loss:
1.5576 - val_accuracy: 0.4634 - val_loss: 2.2665 - learning_rate:
0.0010

Epoch 3: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 3/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 503s 1s/step - accuracy: 0.5203 - loss:
1.4527 - val_accuracy: 0.5253 - val_loss: 2.4270 - learning_rate:
0.0010

Epoch 4: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 4/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 502s 1s/step - accuracy: 0.5431 - loss:
1.4038 - val_accuracy: 0.5625 - val_loss: 1.2846 - learning_rate:
0.0010

Epoch 5: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 5/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 501s 1s/step - accuracy: 0.5695 - loss:
1.2894 - val_accuracy: 0.5768 - val_loss: 1.2100 - learning_rate:
0.0010

Epoch 6: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 6/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 506s 1s/step - accuracy: 0.5821 - loss:
1.2406 - val_accuracy: 0.5487 - val_loss: 1.3606 - learning_rate:
0.0010

Epoch 7: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 7/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 511s 1s/step - accuracy: 0.5875 - loss:
1.2489 - val_accuracy: 0.5877 - val_loss: 1.7394 - learning_rate:
0.0010

Epoch 8: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 8/40
```

```
404/404 ━━━━━━━━━━━━━━━━━━━━ 506s 1s/step - accuracy: 0.5875 - loss:
1.3129 - val_accuracy: 0.5880 - val_loss: 1.2572 - learning_rate:
0.0010

Epoch 9: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 9/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.6003 - loss:
1.2175 - val_accuracy: 0.5944 - val_loss: 1.1801 - learning_rate:
0.0010

Epoch 10: LearningRateScheduler setting learning rate to
0.0010000000474974513.
Epoch 10/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.6115 - loss:
1.1457 - val_accuracy: 0.6130 - val_loss: 1.1368 - learning_rate:
0.0010

Epoch 11: LearningRateScheduler setting learning rate to 0.0001.
Epoch 11/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 511s 1s/step - accuracy: 0.6380 - loss:
1.0738 - val_accuracy: 0.6352 - val_loss: 1.0370 - learning_rate:
1.0000e-04

Epoch 12: LearningRateScheduler setting learning rate to 0.0001.
Epoch 12/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 509s 1s/step - accuracy: 0.6499 - loss:
1.0157 - val_accuracy: 0.6392 - val_loss: 1.0176 - learning_rate:
1.0000e-04

Epoch 13: LearningRateScheduler setting learning rate to 0.0001.
Epoch 13/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 502s 1s/step - accuracy: 0.6597 - loss:
0.9814 - val_accuracy: 0.6402 - val_loss: 1.0083 - learning_rate:
1.0000e-04

Epoch 14: LearningRateScheduler setting learning rate to 0.0001.
Epoch 14/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 497s 1s/step - accuracy: 0.6615 - loss:
0.9756 - val_accuracy: 0.6392 - val_loss: 1.0062 - learning_rate:
1.0000e-04

Epoch 15: LearningRateScheduler setting learning rate to 0.0001.
Epoch 15/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 508s 1s/step - accuracy: 0.6756 - loss:
0.9449 - val_accuracy: 0.6470 - val_loss: 0.9931 - learning_rate:
1.0000e-04

Epoch 16: LearningRateScheduler setting learning rate to 0.0001.
Epoch 16/40
```

```
404/404 ━━━━━━━━━━━━━━━━━━━━ 511s 1s/step - accuracy: 0.6782 - loss:
0.9349 - val_accuracy: 0.6437 - val_loss: 0.9900 - learning_rate:
1.0000e-04

Epoch 17: LearningRateScheduler setting learning rate to 0.0001.
Epoch 17/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 511s 1s/step - accuracy: 0.6775 - loss:
0.9205 - val_accuracy: 0.6476 - val_loss: 0.9911 - learning_rate:
1.0000e-04

Epoch 18: LearningRateScheduler setting learning rate to 0.0001.
Epoch 18/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 512s 1s/step - accuracy: 0.6867 - loss:
0.9142 - val_accuracy: 0.6490 - val_loss: 0.9910 - learning_rate:
1.0000e-04

Epoch 19: LearningRateScheduler setting learning rate to 0.0001.
Epoch 19/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 510s 1s/step - accuracy: 0.6856 - loss:
0.9065 - val_accuracy: 0.6533 - val_loss: 0.9901 - learning_rate:
1.0000e-04

Epoch 20: LearningRateScheduler setting learning rate to 0.0001.
Epoch 20/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 510s 1s/step - accuracy: 0.6914 - loss:
0.8858 - val_accuracy: 0.6454 - val_loss: 0.9949 - learning_rate:
1.0000e-04

Epoch 21: LearningRateScheduler setting learning rate to 1e-05.
Epoch 21/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 511s 1s/step - accuracy: 0.7029 - loss:
0.8691 - val_accuracy: 0.6522 - val_loss: 0.9874 - learning_rate:
1.0000e-05

Epoch 22: LearningRateScheduler setting learning rate to 1e-05.
Epoch 22/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 512s 1s/step - accuracy: 0.7012 - loss:
0.8705 - val_accuracy: 0.6518 - val_loss: 0.9876 - learning_rate:
1.0000e-05

Epoch 23: LearningRateScheduler setting learning rate to 1e-05.
Epoch 23/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 514s 1s/step - accuracy: 0.7024 - loss:
0.8567 - val_accuracy: 0.6529 - val_loss: 0.9878 - learning_rate:
1.0000e-05

Epoch 24: LearningRateScheduler setting learning rate to 1e-05.
Epoch 24/40
404/404 ━━━━━━━━━━━━━━━━━━━━ 513s 1s/step - accuracy: 0.7035 - loss:
```

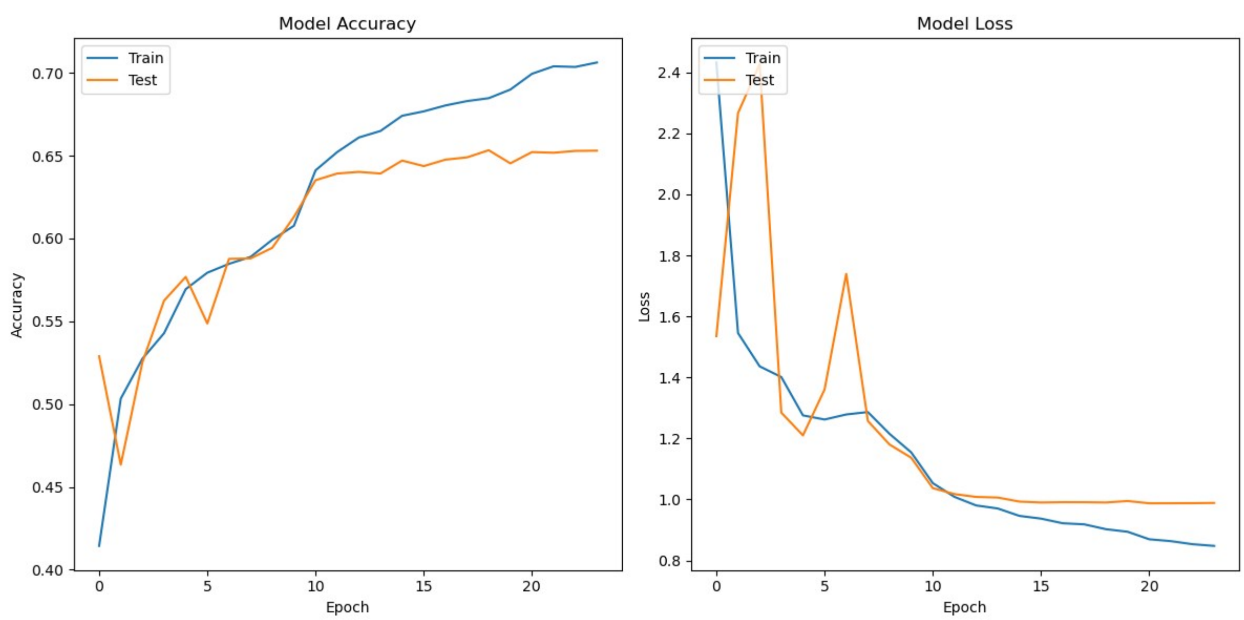
```
0.8502 - val_accuracy: 0.6530 - val_loss: 0.9886 - learning_rate:
1.0000e-05
```

Evaluating ResNet50V2

```python
# Evaluate the model
ResNet50V2_Score = model.evaluate(test_data, steps=len(test_data))

# Print results
print("    Test Loss: {:.5f}".format(ResNet50V2_Score[0]))
print("Test Accuracy: {:.2f}%".format(ResNet50V2_Score[1] * 100))
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 68s 601ms/step - accuracy: 0.5943 - loss:
1.1239
    Test Loss: 0.99010
Test Accuracy: 65.33%
```

```python
plot_curves(history)
```



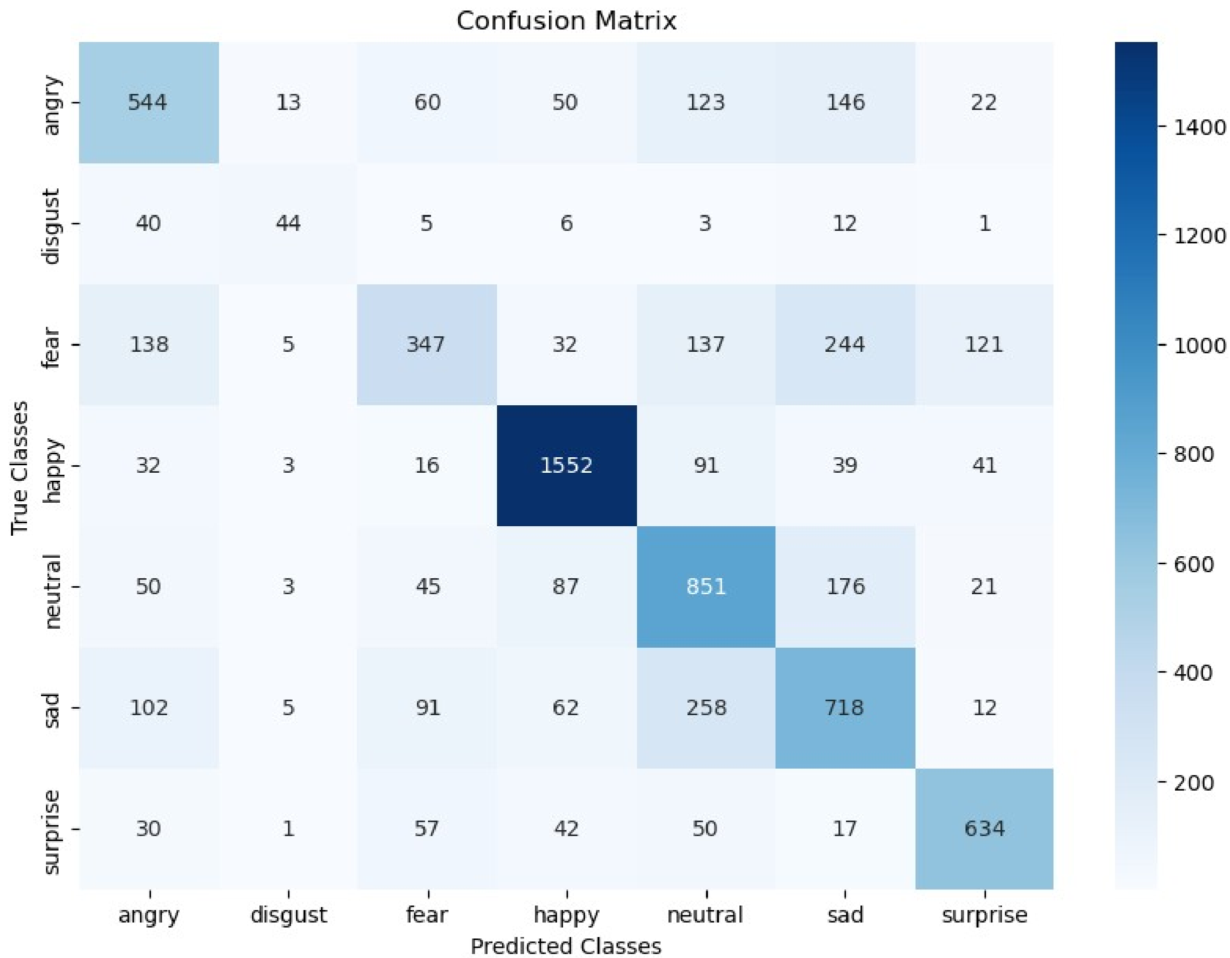

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator

ResNet50V2_Predictions = model.predict(test_data,
steps=len(test_data))
```

```
# Choosing highest probalbilty class in every prediction
ResNet50V2_Predictions = np.argmax(ResNet50V2_Predictions, axis=1)
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 67s 594ms/step
```

```
# Assuming you have test_data and model ready
# Ensure shuffle=False in your test_data if you compare against
test_data.classes

# Predict class probabilities and convert to class predictions
predictions = model.predict(test_data, steps=len(test_data))
predicted_classes = np.argmax(predictions, axis=1)

# Assuming test_data has property 'classes' which contains the true
labels
true_classes = test_data.classes
class_labels = list(test_data.class_indices.keys())  # Getting class
labels from the generator

# Compute the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)

# Plotting the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.ylabel('True Classes')
plt.xlabel('Predicted Classes')
plt.show()
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 67s 592ms/step
```



Visualizing Predictions

```
Emotion_Classes = ['Angry',
                  'Disgust',
                  'Fear',
                  'Happy',
                  'Neutral',
                  'Sad',
                  'Surprise']

# Shuffling Test Data to show diffrent classes
test_preprocessor = ImageDataGenerator(
        rescale = 1 / 255.,
    )

test_generator = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
```

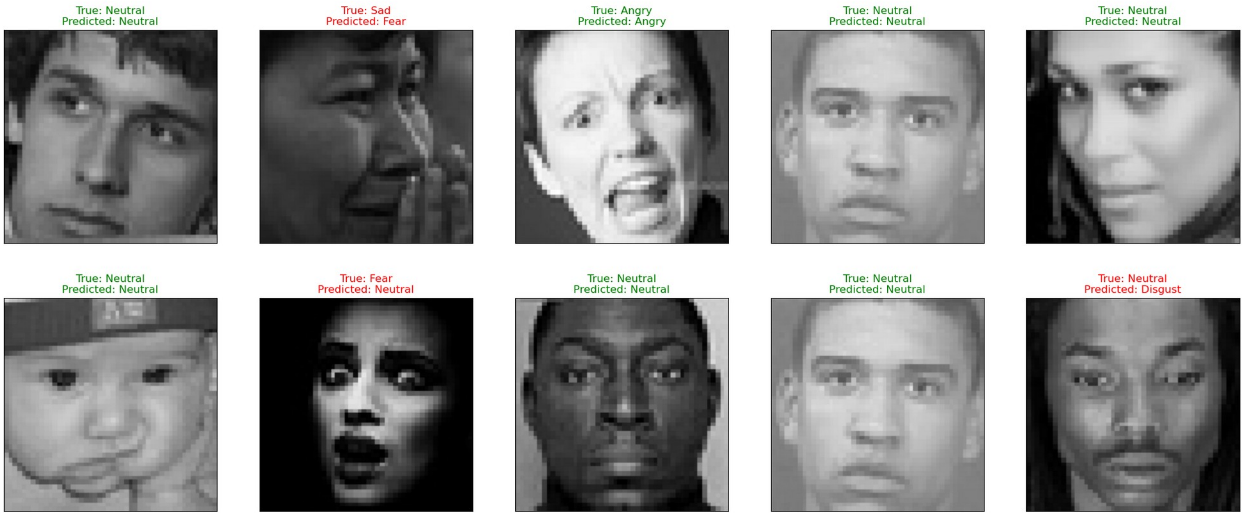
```
    color_mode="rgb",
    shuffle=True,
    batch_size=batch_size,
)
```

```
Found 7179 images belonging to 7 classes.
```

ResNet50V2 Predictions

Displaying model predictions provides direct feedback on the model's current performance but alsoa qualitative tool to see the effectiveness of the training and fine-tuning processes. It is a way to ensure that the model operates correctly and efficiently before deployment.

```
# Display 10 random pictures from the dataset with their labels

Random_batch = np.random.randint(0, len(test_generator) - 1)

Random_Img_Index = np.random.randint(0, batch_size - 1 , 10)

fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(25, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):

    Random_Img = test_generator[Random_batch][0][Random_Img_Index[i]]

    Random_Img_Label = np.argmax(test_generator[Random_batch][1]
[Random_Img_Index[i]])

    Model_Prediction =
np.argmax(ResNet50V2_Model.predict( tf.expand_dims(Random_Img, axis=0)
, verbose=0))

    ax.imshow(Random_Img)

    if Emotion_Classes[Random_Img_Label] ==
Emotion_Classes[Model_Prediction]:
          color = "green"
    else:
          color = "red"
    ax.set_title(f"True: {Emotion_Classes[Random_Img_Label]}\
nPredicted: {Emotion_Classes[Model_Prediction]}", color=color)
plt.show()
plt.tight_layout()
```




```
<Figure size 640x480 with 0 Axes>
```

Music Player

```
Music_Player =
pd.read_csv("~/Downloads/EmotionBasedMusicRecommendationSystem/Emotion
BasedMusicRecommendationSystem/dataset/data_moods.csv")
Music_Player = Music_Player[['name','artist','mood','popularity']]
Music_Player.head()
```

```
                                          name          artist
mood  \
0                                         1999          Prince
Happy
1                                           23  Blonde Redhead
Sad
2                                     9 Crimes     Damien Rice
Sad
3                               99 Luftballons            Nena
Happy
4  A Boy Brushed Red Living In Black And White       Underoath
Energetic

   popularity
0          68
1          43
2          60
3           2
4          60
```

```
Music_Player["mood"].value_counts()
```

```
mood
Sad          197
Calm         195
Energetic    154
Happy        140
Name: count, dtype: int64
```

```
Music_Player["popularity"].value_counts()
```

```
popularity
0     92
51    23
52    22
50    21
55    21
      ..
80     1
2      1
14     1
15     1
88     1
Name: count, Length: 83, dtype: int64
```

```
Play = Music_Player[Music_Player['mood'] =='Calm' ]
Play = Play.sort_values(by="popularity", ascending=False)
Play = Play[:5].reset_index(drop=True)
display(Play)
```

	name	artist	mood	popularity
0	Lost	Annelie	Calm	64
1	Curiosity	Beau Projet	Calm	60
2	Escaping Time	Benjamin Martins	Calm	60
3	Just Look at You	369	Calm	59
4	Vague	Amaranth Cove	Calm	59

```
# Making Songs Recommendations Based on Predicted Class
def Recommend_Songs(pred_class):

    if( pred_class=='Disgust' ):

        Play = Music_Player[Music_Player['mood'] =='Sad' ]
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        display(Play)

    if( pred_class=='Happy' or pred_class=='Sad' ):

        Play = Music_Player[Music_Player['mood'] =='Happy' ]
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        display(Play)
```

```python
    if( pred_class=='Fear' or pred_class=='Angry' ):

        Play = Music_Player[Music_Player['mood'] =='Calm' ]
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        display(Play)

    if( pred_class=='Surprise' or pred_class=='Neutral' ):

        Play = Music_Player[Music_Player['mood'] =='Energetic' ]
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        display(Play)
```

Predicting New Images

```python
# Download Haar Cascade XML file using requests
import requests

url =
"https://raw.githubusercontent.com/opencv/opencv/master/data/haarcasca
des/haarcascade_frontalface_default.xml"
response = requests.get(url)

with open("haarcascade_frontalface_default.xml", "wb") as file:
    file.write(response.content)

# Verify the file is downloaded
import os
if os.path.exists("haarcascade_frontalface_default.xml"):
    print("File downloaded successfully")
else:
    print("Failed to download the file")
```

```
File downloaded successfully
```

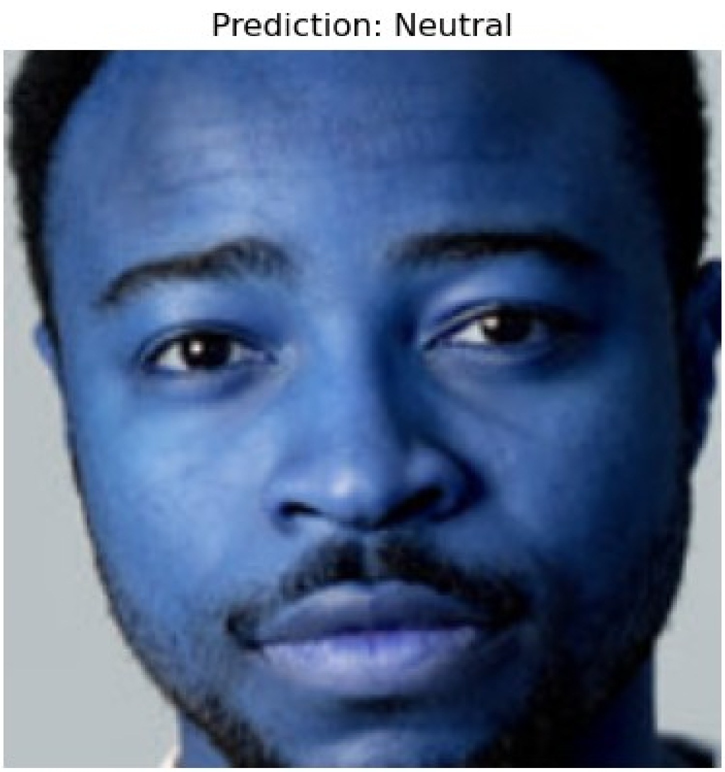
```python
import cv2

# Load the Haar Cascade Classifier
faceCascade =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# Verify that the classifier loaded correctly
if faceCascade.empty():
    print("Failed to load the cascade classifier")
else:
    print("Cascade classifier loaded successfully")
```

```
Cascade classifier loaded successfully
```

```python
def load_and_prep_image(filename, img_shape=224):
    # Expand user directory
    filename = os.path.expanduser(filename)

    # Check if the file exists
    if not os.path.exists(filename):
        raise FileNotFoundError(f"No such file: '{filename}'")

    # Load the image
    img = cv2.imread(filename)

    # Check if the image is loaded correctly
    if img is None:
        raise ValueError(f"Failed to load image from {filename}")

    # Convert to grayscale
    GrayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect faces
    faces = faceCascade.detectMultiScale(GrayImg, 1.1, 4)

    for x, y, w, h in faces:
        img = img[y:y+h, x:x+w]

    # Resize the image
    img = cv2.resize(img, (img_shape, img_shape))
    img = img / 255.0  # Normalize to [0, 1]

    return img

def pred_and_plot(filename, class_names):
    # Import the target image and preprocess it
    img = load_and_prep_image(filename)

    # Make a prediction
    pred = ResNet50V2_Model.predict(np.expand_dims(img, axis=0))

    # Get the predicted class
    pred_class = class_names[np.argmax(pred)]

    # Plot the image with the predicted class
    plt.imshow(img)
    plt.title(f"Prediction: {pred_class}")
    plt.axis(False)
    plt.show()

# Additional Error Handling for FileNotFoundError
try:
```

```
    pred_and_plot(file_path, Emotion_Classes)
except FileNotFoundError as e:
    print(e)
except ValueError as e:
    print(e)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
```



Emotion Detection Music Recommendor

Below is the integration of emotion prediction with music recommendation, enhancing the user experience by personalizing content based on emotional cues. **Music Player Data Load**: Initially, the system loads a dataset (`data_moods.csv`) containing songs and their associated mood metadata. **Emotion Prediction**: The system predicts the emotion from the preprocessed image using the model coded prior. **Image Display**: Showcases the original image alongside the detected emotion, providing immediate visual feedback. **Prediction Display**: Lists the predicted emotion, enhancing user understanding of how the image's emotional content was interpreted by the model. **Recommendations Display**: Presents a list of songs corresponding to the detected emotion, formatted as a table for easy reading.

```
import cv2
import numpy as np
import pandas as pd
```

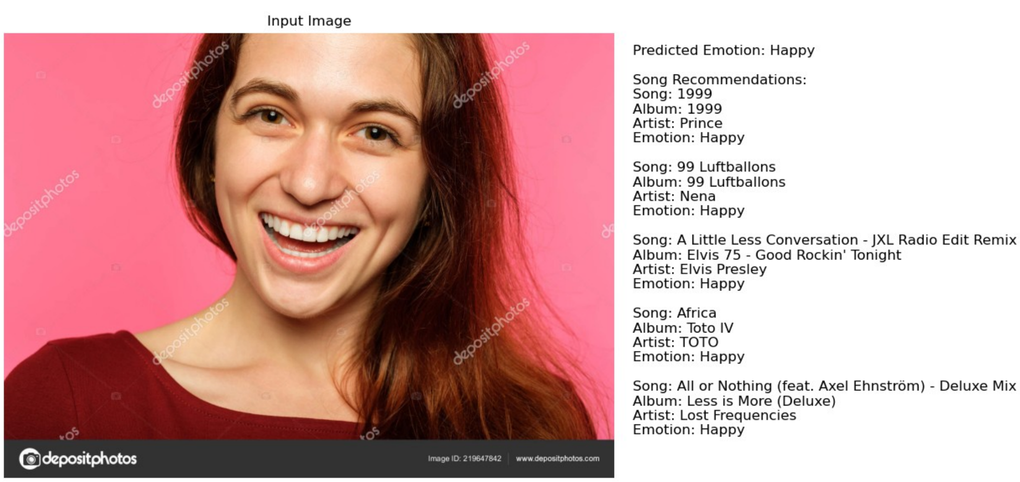
```
# Load the music player data
Music_Player =
pd.read_csv("~/Downloads/EmotionBasedMusicRecommendationSystem/Emotion
BasedMusicRecommendationSystem/dataset/data_moods.csv")

# Preprocess the image for prediction
def load_and_prep_image(filename, img_shape=128):
    img = cv2.imread(filename)
    if img is None:
        raise FileNotFoundError("Image file not found. Check the file
path.")

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (img_shape, img_shape))
    img = img.astype('float32')
    img /= 255.0
    return img

# Predict emotion from an image and recommend songs based on the
emotion
def predict_emotion_and_recommend(filename):
    img = load_and_prep_image(filename)
    pred = model.predict(np.expand_dims(img, axis=0))
    pred_class = Emotion_Classes[np.argmax(pred)]
    recommendations = recommend_songs(pred_class)
    return pred_class, recommendations

# Recommend songs based on the predicted emotion
def recommend_songs(pred_class):
    recommendations = []
    if pred_class == 'Disgust':
        recommendations = get_music_recommendations('Sad')
    elif pred_class in ['Happy', 'Sad']:
        recommendations = get_music_recommendations('Happy')
    elif pred_class in ['Fear', 'Angry']:
        recommendations = get_music_recommendations('Calm')
    elif pred_class in ['Surprise', 'Neutral']:
        recommendations = get_music_recommendations('Energetic')
    return recommendations

# Retrieve music recommendations for a given mood
def get_music_recommendations(mood):
    songs = Music_Player[Music_Player['mood'] == mood]
    songs = songs.sort_values(by="popularity",
ascending=False).head(5)
    recommendations = songs[['album', 'artist', 'name', 'popularity',
'release_date']].to_dict(orient='records')
    return recommendations
```

```
import matplotlib.pyplot as plt
import cv2

def display_results(image_path, predicted_emotion,
song_recommendations):
    # Load and display the image
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Image file not found at
{image_path}. Check the file path.")

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert from BGR to
RGB
    plt.figure(figsize=(15, 8))

    # Plotting the image
    plt.subplot(1, 3, 1)
    plt.imshow(img)
    plt.title('Input Image')
    plt.axis('off')

    # Display the predictions
    plt.subplot(1, 3, 2)
    plt.axis('off')
    plt.title('Prediction Results')
    text = f"Predicted Emotion: {predicted_emotion}\n"
    plt.text(0.01, 0.5, text, fontsize=12, va='center')

    # Create a DataFrame for the song recommendations and display as a
table
    plt.subplot(1, 3, 3)
    plt.axis('off')
    plt.title('Song Recommendations')
    df = pd.DataFrame(song_recommendations)
    cell_text = []
    for row in range(len(df)):
        cell_text.append(df.iloc[row])
    table = plt.table(cellText=cell_text, colLabels=df.columns,
cellLoc = 'center', loc='center', colColours
=["palegreen"]*df.shape[1])
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1.2, 1.2)


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

Demo

```
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_and_prep_image(filename, img_shape=128):
    """ Load and prepare an image for model prediction. """
    img = cv2.imread(filename)
    if img is None:
        raise FileNotFoundError(f"Image file not found: {filename}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (img_shape, img_shape))
    img = img.astype('float32') / 255.0
    return img

def predict_emotion(image_path):
    """ Predict the emotion from an image. """
    img = load_and_prep_image(image_path)
    pred = model.predict(np.expand_dims(img, axis=0))
    pred_class = Emotion_Classes[np.argmax(pred)]
    return pred_class

def get_song_recommendations(emotion):
    """ Get song recommendations based on the predicted emotion. """
    if emotion == 'Disgust':
        filter_mood = 'Sad'
    elif emotion in ['Happy', 'Sad']:
        filter_mood = 'Happy'
    elif emotion in ['Fear', 'Angry']:
        filter_mood = 'Calm'
    elif emotion in ['Surprise', 'Neutral']:
        filter_mood = 'Energetic'

    songs = Music_Player[Music_Player['mood'] == filter_mood]
    return songs[['name', 'album', 'artist',
'mood']].head(5).to_dict(orient='records')

def display_results(image_path, predicted_emotion,
song_recommendations):
    """ Display the image, predicted emotion, and song
recommendations. """
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    plt.figure(figsize=(15, 8))

    # Display the input image
    plt.subplot(1, 2, 1)
```

```
    plt.imshow(img)
    plt.title('Input Image')
    plt.axis('off')

    # Display the prediction and recommendations
    plt.subplot(1, 2, 2)
    plt.axis('off')
    text = f"Predicted Emotion: {predicted_emotion}\n\nSong
Recommendations:\n"
    for song in song_recommendations:
        song_info = f"Song: {song['name']}\nAlbum: {song['album']}\
nArtist: {song['artist']}\nEmotion: {song['mood']}\n"
        text += song_info + "\n"

    plt.text(0.01, 0.5, text, fontsize=12, va='center', ha='left')
    plt.tight_layout()
    plt.show()


# Example usage
image_path = r'C:\Users\Manahil\Downloads\
EmotionBasedMusicRecommendationSystem\
EmotionBasedMusicRecommendationSystem/uploads/happy image rec
test.jpg'  # Update with the actual path to your image
predicted_emotion = predict_emotion(image_path)
song_recommendations = get_song_recommendations(predicted_emotion)
display_results(image_path, predicted_emotion, song_recommendations)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```



```python
# Example usage
image_path = r'C:\Users\Manahil\Downloads\
EmotionBasedMusicRecommendationSystem\
EmotionBasedMusicRecommendationSystem/uploads/sadimg.jpg'  # Update
with the actual path to your image
predicted_emotion = predict_emotion(image_path)
song_recommendations = get_song_recommendations(predicted_emotion)
display_results(image_path, predicted_emotion, song_recommendations)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```


```python
# Example usage
image_path = r'C:\Users\Manahil\Downloads\
EmotionBasedMusicRecommendationSystem\
EmotionBasedMusicRecommendationSystem/uploads/fear.jpg'  # Update with
the actual path to your image
predicted_emotion = predict_emotion(image_path)
song_recommendations = get_song_recommendations(predicted_emotion)
display_results(image_path, predicted_emotion, song_recommendations)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

Input Image

Predicted Emotion: Fear

Song Recommendations:
Song: A Burden to Bear
Album: A Burden to Bear
Artist: Emmanuelle Rimbaud
Emotion: Calm

Song: A La Plage
Album: A La Plage
Artist: Ron Adelaar
Emotion: Calm

Song: Adjustments
Album: Adjustments
Artist: Josie Mehlin
Emotion: Calm

Song: Adrift
Album: Adrift
Artist: Cooper Sams
Emotion: Calm

Song: After The Rain
Album: After The Rain
Artist: Comet Blue
Emotion: Calm

```
# Example usage
image_path = r'C:\Users\Manahil\Downloads\
EmotionBasedMusicRecommendationSystem\
EmotionBasedMusicRecommendationSystem/uploads/angry image rec
test.jpg'  # Update with the actual path to your image
predicted_emotion = predict_emotion(image_path)
song_recommendations = get_song_recommendations(predicted_emotion)
display_results(image_path, predicted_emotion, song_recommendations)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

Input Image

Predicted Emotion: Angry

Song Recommendations:
Song: A Burden to Bear
Album: A Burden to Bear
Artist: Emmanuelle Rimbaud
Emotion: Calm

Song: A La Plage
Album: A La Plage
Artist: Ron Adelaar
Emotion: Calm

Song: Adjustments
Album: Adjustments
Artist: Josie Mehlin
Emotion: Calm

Song: Adrift
Album: Adrift
Artist: Cooper Sams
Emotion: Calm

Song: After The Rain
Album: After The Rain
Artist: Comet Blue
Emotion: Calm