Configuration Manual

MSc Research Project
Data Analytics

Vipin Sharma
Student ID: 22207406

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Vipin Sharma		
Student ID:	X22207406		
Programme:	Data Analytics	**Year:**	2023-24
Module:	MSc Research Project		
Lecturer:	Ahmed Makki		
Submission Due Date:	12/08/2024		
Project Title:	Improving Emotion Detection and Music Recommendation Through Advanced Facial Recognition and Optimized Hyper-parameters Tuning		
Word Count:	879	**Page Count: 15**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vipin Sharma
Date: 11/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

CONFIGURATION MANUAL

VIPIN SHARMA

Student ID: X22207406

1 INTRODUCTION

The steps required to finish the research project, Improving Emotion Detection and Music Recommendation Through Advanced Facial Recognition and Optimized Hyperparameters Tuning (1) are provided in this configuration manual along with the system configuration, software, and hardware requirements information.

The structure of the configuration manual is as: In the section 2 all the information regarding the software and hardware are mentioned which is used for this research. The library which is imported, basic data visualization, and train-test splitting of the data are covered in the section 3. Section 4 shows the image pre-processing steps of the data for the different architectures and advanced pre-trained models. In section 5 shows the CNN architecture, ResNet50, and Xception model architectures with hyper-parameter tuning. The predictions of the images are shown by using the different models in section 6. In section 7, the Music Player UDF, basic visualization, and basic pre-processing of song data are covered. In the last section, 8 Finally, recommendations of songs on the new images have been shown. In the last Referencing are mentioned.

2 SYSTEM CONFIGURATION

The specifications for the hardware and software required for this project are given in the section below.

2.1 Hardware Requirements

System Name	Asus VivoBook 15
Operating System	Windows 11
RAM	8.00 GB
Hard Disc Space	477GB
CPU	64-bit, i5-1235U CPU @1300Mhz

Table 1: Hardware Requirements

2.2 Software Requirements

Programming Language Tools	Visual Studio Code
Web Browser	Google Chrome
Other Software's	Overleaf, Microsoft Word

Table 2: Software Requirements

3 PROJECT DEVELOPMENT

In this section Python library that is used for this project, basic data visualization, and train-test splitting of the data are covered.

3.1 Python Library

```
#Import all the library

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

import os
import pickle
import tensorflow as tf
import keras
import cv2
import urllib.request

from tensorflow.keras.preprocessing import image
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import load_model


from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

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 import *
from tensorflow.keras.applications import VGG16, ResNet50
```

Figure 1: Python Library

3.2 Train-Test Splitting

```python
train_data = r'D:\\NCI Documents\\NCI Semester 3\\Recommenation songs using facial expression\\Face Dataset\\train'
test_data = r'D:\\NCI Documents\\NCI Semester 3\\Recommenation songs using facial expression\\Face Dataset\\test'

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

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

    df = pd.DataFrame.from_dict(Classes_Dict, orient='index', columns=[name])
    return df

# Get class counts for train and test datasets
Train_Count = Classes_Count(train_data, "Train").sort_values(by='Train', ascending=False)
Test_Count = Classes_Count(test_data, "Test").sort_values(by='Test', ascending=False)
```

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

✓ 0.0s

	Train	Test
happy	7215	1774
neutral	4965	1233
sad	4830	1247
fear	4097	1024
angry	3995	959
surprise	3171	831
disgust	436	111

Figure 2: Train-Test Splitting

3.3 Basic Visualizations

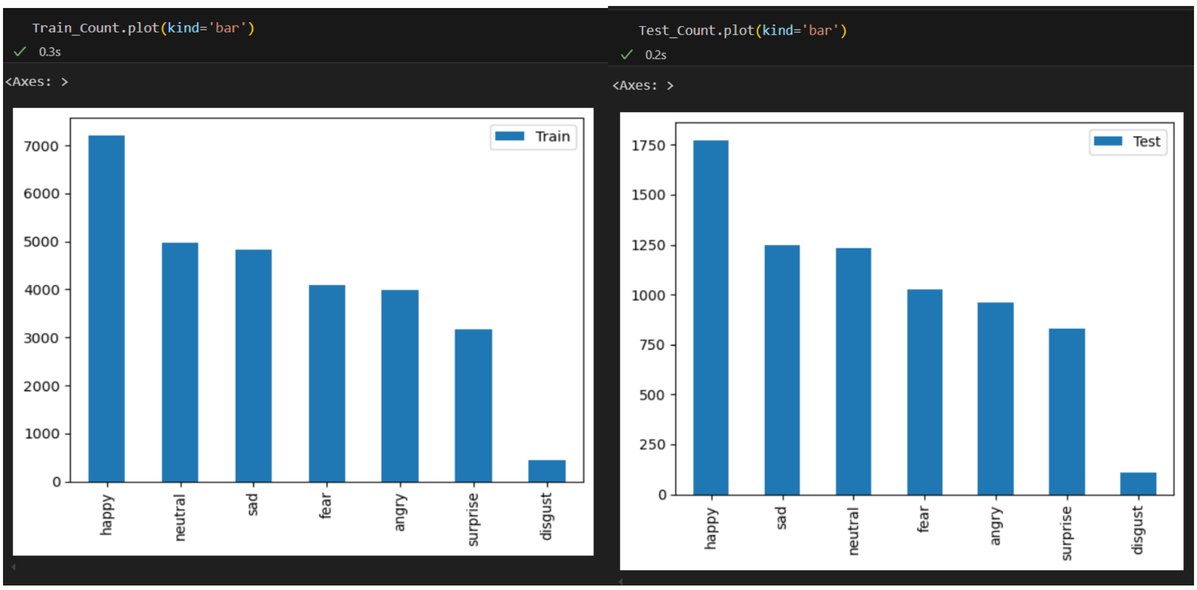


Figure 3: Train-Test Visualization

```python
plt.style.use('default')
plt.figure(figsize=(25, 8))

image_count = 1
BASE_URL = r'D:\NCI Documents\NCI Semester 3\Recommenation songs using facial expression\Face Dataset\train'  # Use raw string

for directory in os.listdir(BASE_URL):
    dir_path = os.path.join(BASE_URL, directory)  # Join paths correctly
    if os.path.isdir(dir_path):  # Ensure it's a directory
        for i, file in enumerate(os.listdir(dir_path)):
            if i == 1:
                break
            else:
                fig = plt.subplot(1, 7, image_count)
                image_count += 1
                file_path = os.path.join(dir_path, file)  # Join paths correctly
                image = cv2.imread(file_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert color for correct display
                plt.imshow(image)
                plt.title(directory, fontsize=20)
                plt.axis('off')  # Hide axes for better visualization

plt.show()
```

✓ 0.5s Python

Figure 4: Different Types of Facial Expression

4 IMAGE PROCESSING

The Pre-processing of the data for every architecture CNN with hyper-parameter and two advanced pre-trained deep learning models with hyper-parameter ResNet50 and Xception are shown in this section. To learn and understand the pre-trained model better the author read the ResNet50 article (Ruiz, April 2024)(15) and for the Xception read the (Sarkar, May 2019)(15).

4.1 Pre-processing and augmentation for CNN Model

```
img_shape = (48,48)
batch_size = 64
# Define data preprocessors
#Data Augementation
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.
)

# Load train and test data
trained_data = train_preprocessor.flow_from_directory(
    train_data,
    class_mode='categorical',
    target_size=(img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training'
)
```

Figure 5: CNN Pre- Processing

4.2 Pre-processing and augmentation for ResNet50 Model

ResNet50 Model

```
#Specifyinng the New image shape for the resnet
img_shape = 224
batch_size=64
```

✓ 0.0s

```
# Define data preprocessors
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.
)
# Load train and test data
trained_data = train_preprocessor.flow_from_directory(
    train_data,
    class_mode='categorical',
    target_size=(img_shape, img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training'
)

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

✓ 1.4s

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

Figure 5: ResNet50 Pre-Processing

4.3 Pre-processing and augmentation for Xception Model

Xception Model

```
# Define the new image shape for the Xception model
img_shape = 224
batch_size = 64
```

✓ 0.0s

```
# Define data preprocessors
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.
)

# Load train and test data
trained_data = train_preprocessor.flow_from_directory(
    train_data,
    class_mode='categorical',
    target_size=(img_shape, img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training'
)
tested_data = test_preprocessor.flow_from_directory(
    test_data,
    class_mode='categorical',
    target_size=(img_shape, img_shape),
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)
```

✓ 1.4s

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

Figure 6: Xception Pre-Processing

5 MODEL ARCHITECTURES

5.1 CNN Model Architectures

Creating own CNN Model Architecture

```
def Convoltional_Neural_network():

    # Create a Sequential model
    model = Sequential()

    # Add three convolutional layers with max pooling

    #CNN1
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(img_shape[0],img_shape[1],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, kernel_size=(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, kernel_size=(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))
model.add(Flatten())

# Add a fully connected layer with dropout
model.add(Dense(1024, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Add a fully connected layer with dropout
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Add a fully connected layer with dropout
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Add a fully connected layer with dropout
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Add a fully connected layer with dropout
model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Add a fully connected layer with dropout
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

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

return model
```

```
CNN_Model = Convoltional_Neural_network()

#Print model summary
CNN_Model.summary()

#Compile the model
CNN_Model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
✓ 1.6s                                                                                                                    Pyth

WARNING:tensorflow:From c:\Users\415vi\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Pleas

WARNING:tensorflow:From c:\Users\415vi\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fu

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 32)        896

 batch_normalization (Batch  (None, 46, 46, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 46, 46, 64)        18496

 batch_normalization_1 (Bat  (None, 46, 46, 64)        256
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 23, 23, 64)        0
 D)

 dropout (Dropout)           (None, 23, 23, 64)        0

 conv2d_2 (Conv2D)           (None, 21, 21, 64)        36928

...
Non-trainable params: 5376 (21.00 KB)
```

```
CNN_final_Score = CNN_Model.evaluate(tested_data)

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

```
✓ 13.7s

WARNING:tensorflow:From c:\Users\415vi\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is

113/113 [==============================] - 14s 113ms/step - loss: 0.9550 - accuracy: 0.6497
Test Loss :0.95499
Test Accuracy:64.97%
```

Figure 7: CNN Model Architecture and Accuracy

5.2 ResNet 50 Model Architectures

```
def Create_resNet50_model():

    model = Sequential([
        ResNet50,
        Dropout(.25),
        BatchNormalization(),
        Flatten(),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(.5),
        Dense(7, activation='softmax')
    ])
    return model
```

```
ResNet50_Model= Create_resNet50_model()

ResNet50_Model.summary()

ResNet50_Model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```
✓ 0.7s

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 dropout_9 (Dropout)         (None, 7, 7, 2048)        0

 batch_normalization_12 (Ba  (None, 7, 7, 2048)        8192
 tchNormalization)

 flatten_1 (Flatten)         (None, 100352)            0

 dense_7 (Dense)             (None, 64)                6422592

 batch_normalization_13 (Ba  (None, 64)                256
 tchNormalization)

 dropout_10 (Dropout)        (None, 64)                0

 dense_8 (Dense)             (None, 7)                 455

=================================================================
Total params: 30019207 (114.51 MB)
Trainable params: 23377799 (89.18 MB)
Non-trainable params: 6641408 (25.33 MB)
```

```
ResNet50_final_Score = ResNet50_Model.evaluate(tested_data)

print("Test Loss :{:.5f}".format(ResNet50_final_Score[0]))
print("Test Accuracy:{:.2f}%".format(ResNet50_final_Score[1]*100))
```
✓ 6m 22.3s

```
113/113 [==============================] - 382s 3s/step - loss: 1.2577 - accuracy: 0.5175
Test Loss :1.25774
Test Accuracy:51.75%
```

Figure 8: ResNet50 Model Architecture and Accuracy

5.3 Xception Model Architectures

```
# Create the Xception-based model
def Create_Xception_model():
    model = Sequential([
        Xception_base,
        Dropout(0.25),
        BatchNormalization(),
        Flatten(),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(7, activation='softmax')
    ])
    return model
```
✓ 0.0s

```
Xception_Model = Create_Xception_model()
Xception_Model.summary()
Xception_Model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```
✓ 0.7s

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 7, 7, 2048)        20861480

 dropout_11 (Dropout)        (None, 7, 7, 2048)        0

 batch_normalization_18 (Ba  (None, 7, 7, 2048)        8192
 tchNormalization)

 flatten_2 (Flatten)         (None, 100352)            0

 dense_9 (Dense)             (None, 64)                6422592

 batch_normalization_19 (Ba  (None, 64)                256
 tchNormalization)

 dropout_12 (Dropout)        (None, 64)                0

 dense_10 (Dense)            (None, 7)                 455

=================================================================
Total params: 27292975 (104.11 MB)
Trainable params: 18595575 (70.94 MB)
Non-trainable params: 8697400 (33.18 MB)
_________________________________________________________________

# Load the model
Xception_Model = load_model("Xception_Model_25.h5")

✓ 4.8s
+ Code   + Markdown

# Evaluate the Xception model on the test data
Xception_Score = Xception_Model.evaluate(tested_data)

# Print the test loss and accuracy
print("Test Loss :{:.5f}".format(Xception_Score[0]))
print("Test Accuracy:{:.2f}%".format(Xception_Score[1]*100))
✓ 6m 46.3s

113/113 [==============================] - 406s 4s/step - loss: 1.0119 - accuracy: 0.7023
Test Loss :1.01191
Test Accuracy:70.23%
```

Figure 9: Xception Model Architecture and Accuracy

6 IMAGE PREDICTION USING MODELS

6.1 CNN Model Prediction

```
CNN Prediction

# Get a random batch index
Random_batch = np.random.randint(0, len(test_generator) - 1)
# Get 10 random image indices within the batch
Random_Img_Index = np.random.randint(0, test_generator.batch_size - 1, 10)

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

# Loop over the 10 random images
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]])

    # Resize the image to the expected input shape of the model
    resized_img = tf.image.resize(Random_Img, (48, 48))

    # Get model prediction
    Model_Prediction = np.argmax(CNN_Model.predict(tf.expand_dims(resized_img, axis=0), verbose=0))

    # Display the image
    ax.imshow(Random_Img)

    # Set title color based on prediction correctness
    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.tight_layout()
plt.show()
```

6.2 CNN Model Prediction

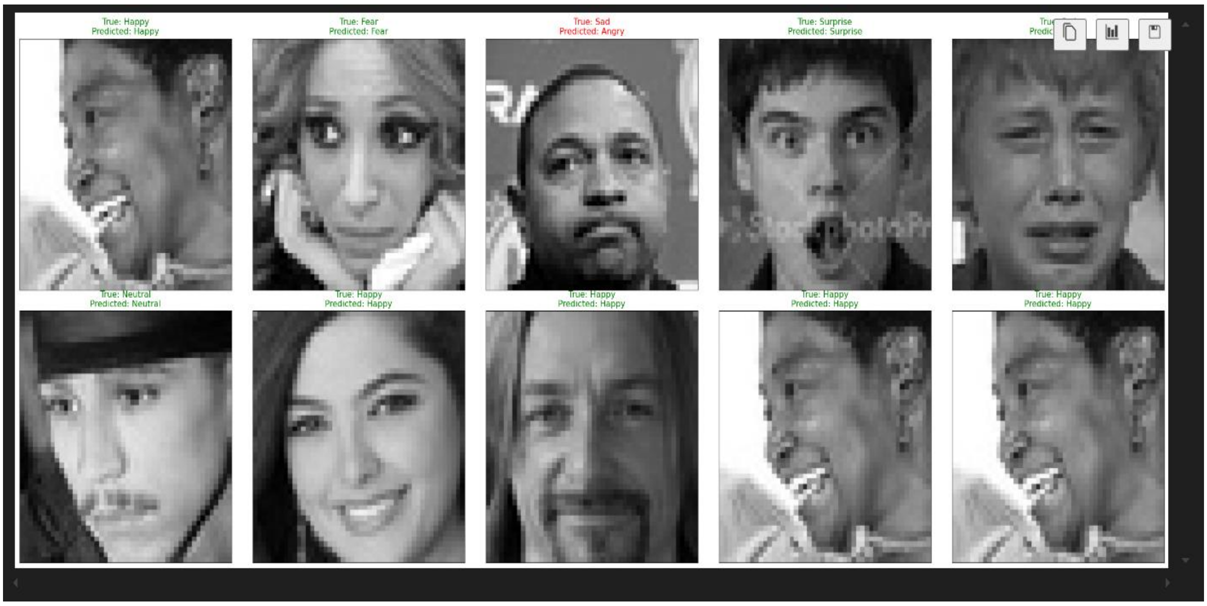


Figure 10: Prediction on different Face Expression

6.3 Xception Model Prediction

Xception Model Prediction

```python
from tensorflow.keras.models import load_model

# Get a random batch index
Random_batch = np.random.randint(0, len(test_generator) - 1)
# Get 10 random image indices within the batch
Random_Img_Index = np.random.randint(0, test_generator.batch_size - 1, 10)
# Set up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(25, 10), subplot_kw={'xticks': [], 'yticks': []})

# Loop over the 10 random images
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]])

    # Resize the image to the expected input shape of the model
    resized_img = tf.image.resize(Random_Img, (224, 224))
    # Get model prediction
    Model_Prediction = np.argmax(Xception_Model.predict(tf.expand_dims(resized_img, axis=0), verbose=0))

    # Display the image
    ax.imshow(Random_Img)
    # Set title color based on prediction correctness
    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.tight_layout()
plt.show()
```

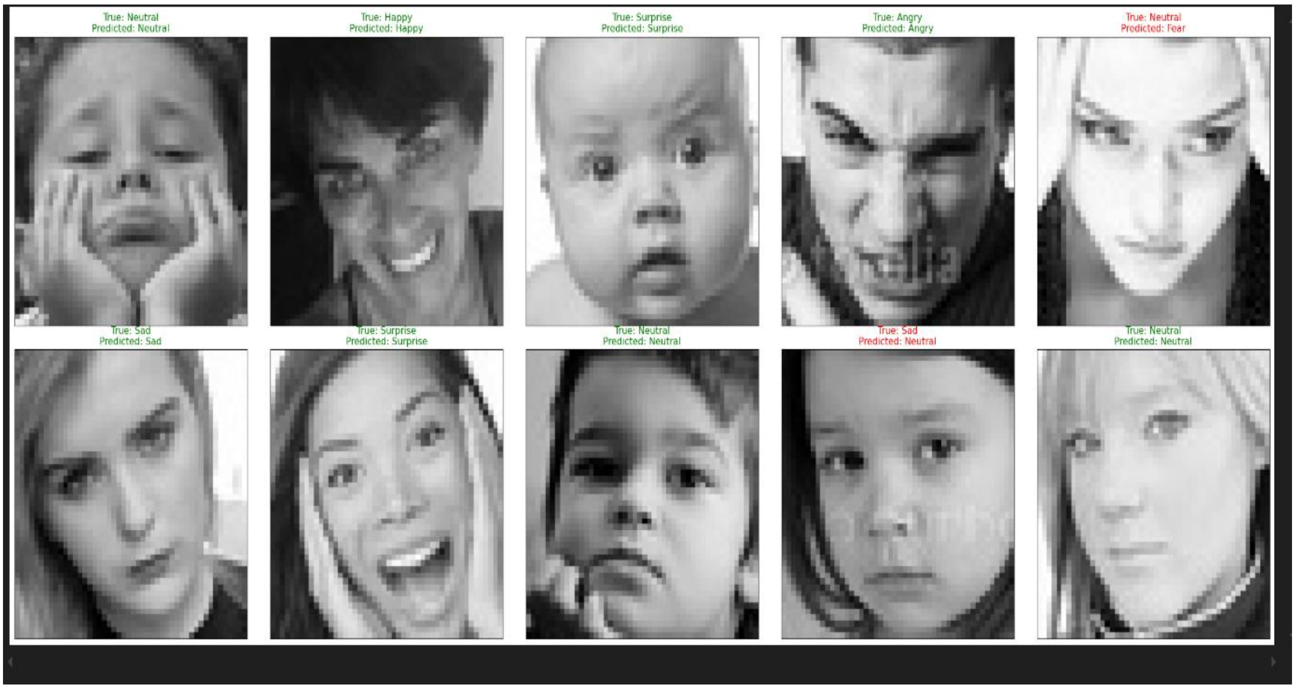


Figure 11: Prediction on different Face Expression

7 MUSIC DATASET LOADING, VISUALIZATION AND UDF

7.1 Working With Music Dataset

```
Music_Player = pd.read_csv(r'D:\NCI Documents\NCI Semester 3\Recommenation songs using facial expression\Face Dataset\Songs Dataset.csv')
Music_Player = Music_Player[['name','artist','mood','popularity']]
Music_Player.head()
```

✓ 0.1s

	name	artist	mood	popularity
0	1999	Prince	Happy	68
1	23	Blonde Redhead	Sad	43
2	9 Crimes	Damien Rice	Sad	60
3	99 Luftballons	Nena	Happy	2
4	A Boy Brushed Red Living In Black And White	Underoath	Energetic	60

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

✓ 0.0s

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

Figure 12: Load the Music Data

7.2 Pre-Processing and Understanding

```python
Play = Music_Player[Music_Player['mood']=='Calm']
Play = Play.sort_values(by='popularity', ascending=False)
Play = Play[:7].reset_index(drop=True)
display(Play)
```

✓ 0.0s

	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
5	What You Love You Must Love Now	The Six Parts Seven	Calm	59
6	alpha waves	Eucalyptic	Calm	59

Figure 13: Preprocessing and Sorting

7.3 Create User-Define Function (UDF)

```python
#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)

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

        Play = Music_Player[Music_Player['mood']=='Clam']
        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)
```

✓ 0.0s

Figure 14: Creating User Define Function (UDF)

8 TESTING AND FINAL PREDICTION ON FACIAL IMAGE WITH SONGS

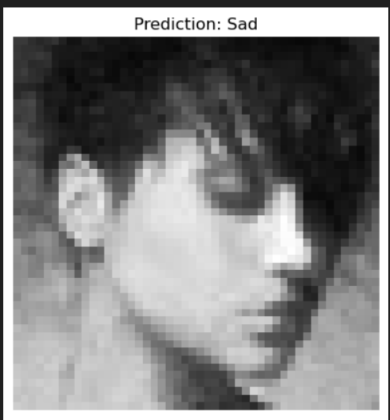
8.1 Prediction by using the Proposed CNN models

Predicting New Images

```python
# Define a function to load and preprocess the image : CNN MOdel
def load_and_prep_image(filename, img_shape=48):
    img = image.load_img(filename, target_size=(img_shape, img_shape))
    img = image.img_to_array(img)
    img = img / 255.0
    return img

# Define the function to predict and plot the result
def pred_and_plot(filename, class_names):
    # Import the target image and preprocess it
    img = load_and_prep_image(filename)

    # Make a prediction using the CNN model
    pred = CNN_Model.predict(np.expand_dims(img, axis=0))

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

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

    # Recommend songs based on the predicted class (assuming you have this function)
    Recommend_Songs(pred_class)

# Assuming 'Emotion_Classes' is defined
Emotion_Classes = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
# Assuming 'CNN_Model' is already loaded and defined
# Example function call
pred_and_plot(r'D:\NCI Documents\NCI Semester 3\Recommenation songs using facial expression\Face Dataset\test\sad\PrivateTest_831970.jpg', Emotion_Classes)
```

Figure 15: Prediction New Images Using CNN Model

```
1/1 [==============================] - 0s 30ms/step
```



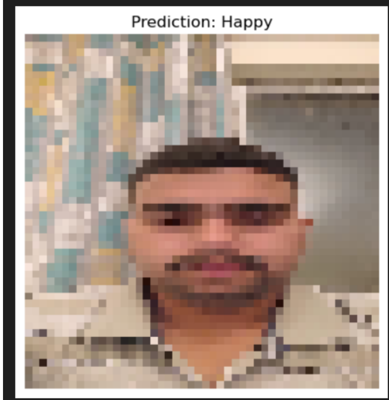
	name	artist	mood	popularity
0	Pumped Up Kicks	Foster The People	Happy	84
1	Africa	TOTO	Happy	84
2	Take on Me	a-ha	Happy	84
3	Highway to Hell	AC/DC	Happy	83
4	Here Comes The Sun - Remastered 2009	The Beatles	Happy	83

Figure 16: Music Recommendation on Predicting Facial Expression

```
pred_and_plot(r'D:\NCI Documents\NCI Semester 3\Recommenation songs using facial expression\WhatsApp Image 2024-07-17 at 22.52.07_1a631c03.jpg', Emotion_Classe
```

✓ 0.2s Python

```
1/1 [==============================] - 0s 26ms/step
```



	name	artist	mood	popularity
0	Pumped Up Kicks	Foster The People	Happy	84
1	Africa	TOTO	Happy	84
2	Take on Me	a-ha	Happy	84
3	Highway to Hell	AC/DC	Happy	83
4	Here Comes The Sun - Remastered 2009	The Beatles	Happy	83

Figure 17: Prediction Facial Expression and Give the Music

9 Referencing

1. Sharma, V., 2024. *Report: Improving Emotion Detection and Music Recommendation Through Advanced Facial Recognition and Optimized Hyper-parameters Tuning*. National College of Ireland.

2. Ruiz, P. (2024, April 30). Understanding and visualizing ResNets - Towards Data Science. *Medium*. Retrieved from https://towardsdatascience.com

3. Sarkar, A. (2023, May 19). Xception: Implementing from scratch using Tensorflow. *Medium*. Retrieved from https://towardsdatascience.com