DEEP LEARNING FINAL PRACTICAL EXAMINATION

Emosic: Emotion-Driven Music Selection

Group Members:

- Sunit Trivedi - 0120200200
- Nehaal Pandey - 0120200245

Batch : DL2

About Dataset

The dataset being used is the **FER 2013 Dataset**.

This Dataset contains **28709** images for training with 3 columns. The data consists of 48x48 pixel grayscale images of faces.

The feature columns include the pixels of human faces of an image and the target column contains the class of emotion.

Following are the seven categories in the dataset:

0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral

Importing dependencies

```
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
```

Importing dataset and Visualizing Classes

[ ] ↳ *5 cells hidden*

Data Preprocessing

```
img_shape = 48
batch_size = 64
train_data_path = '../input/fer2013/train/'
test_data_path = '../input/fer2013/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 7178 images belonging to 7 classes.
```

Building CNN Model

```
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'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 46, 46, 32)        896
_________________________________________________________________
batch_normalization_24 (Batc (None, 46, 46, 32)        128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 46, 46, 64)        18496
_________________________________________________________________
batch_normalization_25 (Batc (None, 46, 46, 64)        256
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_18 (Dropout)         (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 21, 21, 64)        36928
_________________________________________________________________
batch_normalization_26 (Batc (None, 21, 21, 64)        256
```

```
conv2d_15 (Conv2D)           (None, 21, 21, 128)       73856

batch_normalization_27 (Batc (None, 21, 21, 128)       512

max_pooling2d_7 (MaxPooling2 (None, 11, 11, 128)       0

dropout_19 (Dropout)         (None, 11, 11, 128)       0

conv2d_16 (Conv2D)           (None, 9, 9, 128)         147584
```

Specifying Callbacks

```
# Create Callback Checkpoint
checkpoint_path = "CNN_Model_Checkpoint"

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

# 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


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

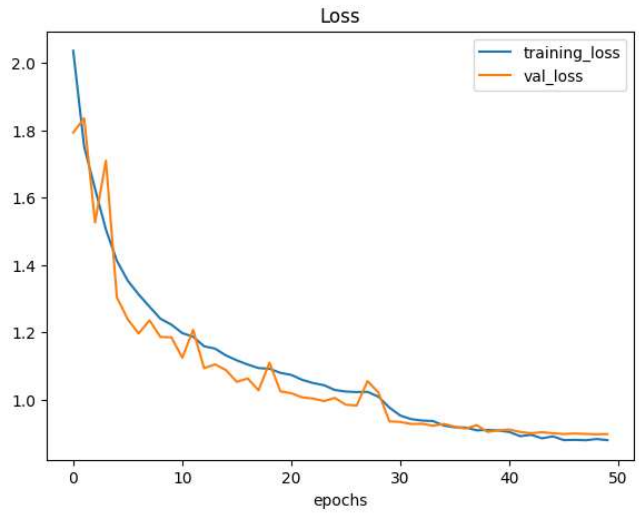
```
Epoch 41/50

448/448 [==============================] - 57s 128ms/step - loss: 0.9039 - accuracy:


Epoch 00041: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.

Epoch 42/50

448/448 [==============================] - 54s 121ms/step - loss: 0.8913 - accuracy:

Epoch 43/50

448/448 [==============================] - 54s 121ms/step - loss: 0.8955 - accuracy:

Epoch 44/50

448/448 [==============================] - 53s 119ms/step - loss: 0.8850 - accuracy:

Epoch 45/50

448/448 [==============================] - 56s 124ms/step - loss: 0.8909 - accuracy:


Epoch 00045: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.

Epoch 46/50

448/448 [==============================] - 53s 119ms/step - loss: 0.8797 - accuracy:

Epoch 47/50

448/448 [==============================] - 54s 120ms/step - loss: 0.8805 - accuracy:

Epoch 48/50

448/448 [==============================] - 56s 124ms/step - loss: 0.8793 - accuracy:


Epoch 00048: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.

Epoch 49/50

448/448 [==============================] - 55s 124ms/step - loss: 0.8830 - accuracy:

Epoch 50/50

448/448 [==============================] - 54s 120ms/step - loss: 0.8796 - accuracy:
```

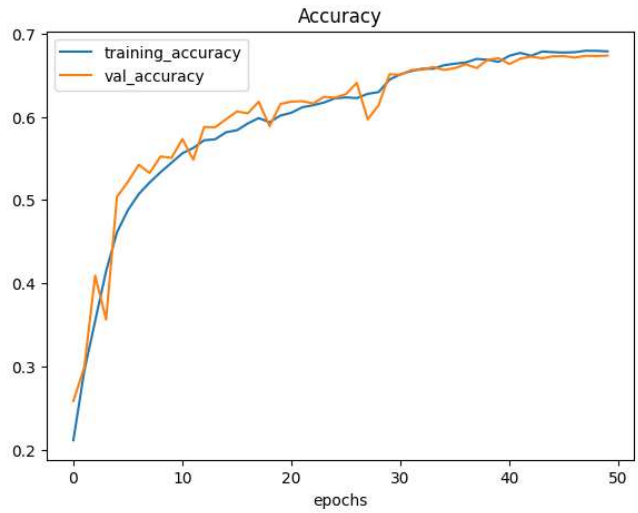
Evaluating CNN Model

```
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 [==============================] - 7s 59ms/step - loss: 0.8965 - accuracy: 0.674

    Test Loss: 0.89649

 Test Accuracy: 67.40%
```

```
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)
```

```
test_data.class_indices
```

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

```
#confusion matrix
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

Data Preprocessing

```
# specifing new image shape for resnet
img_shape = 224
batch_size = 64
train_data_path = '../input/fer2013/train/'
test_data_path = '../input/fer2013/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 7178 images belonging to 7 classes.
```

Fine-Tuning ResNet50V2

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

#ResNet50V2.summary()
```

```
# Freezing all layers except last 50

ResNet50V2.trainable = True

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

Building ResNet50V2 Model

```
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=['accura
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800


 dropout_5 (Dropout)         (None, 7, 7, 2048)        0


 batch_normalization_5 (Batc  (None, 7, 7, 2048)       8192
 hNormalization)
```

```
flatten_2 (Flatten)         (None, 100352)            0

dense_5 (Dense)             (None, 64)                6422592

batch_normalization_6 (Batc  (None, 64)               256
hNormalization)

dropout_6 (Dropout)         (None, 64)                0

dense_6 (Dense)             (None, 7)                 455

=================================================================
Total params: 29,996,295
Trainable params: 22,779,527
Non-trainable params: 7,216,768
_________________________________________________________________
```

Specifying Callbacks

```
# Create Callback Checkpoint
checkpoint_path = "ResNet50V2_Model_Checkpoint"

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

# Create ReduceLROnPlateau Callback to reduce overfitting by decreasing learning
Reducing_LR = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                                   factor=0.2,
                                                   patience=2,
#                                                    min_lr=0.00005,
                                                   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


ResNet50V2_history = ResNet50V2_Model.fit(train_data ,validation_data = test_data , epochs=3
                                          callbacks = callbacks, steps_per_epoch=steps_per_ep
```

```
Epoch 1/30

448/448 [==============================] - 335s 739ms/step - loss: 1.5019 - accuracy

Epoch 2/30

448/448 [==============================] - 328s 732ms/step - loss: 1.2341 - accuracy

Epoch 3/30

448/448 [==============================] - 328s 732ms/step - loss: 1.1402 - accuracy

Epoch 4/30

448/448 [==============================] - 325s 725ms/step - loss: 1.0697 - accuracy

Epoch 5/30

448/448 [==============================] - 328s 733ms/step - loss: 1.0350 - accuracy

Epoch 6/30

448/448 [==============================] - 335s 747ms/step - loss: 1.0101 - accuracy

Epoch 7/30

448/448 [==============================] - 333s 743ms/step - loss: 0.9781 - accuracy

Epoch 8/30
```

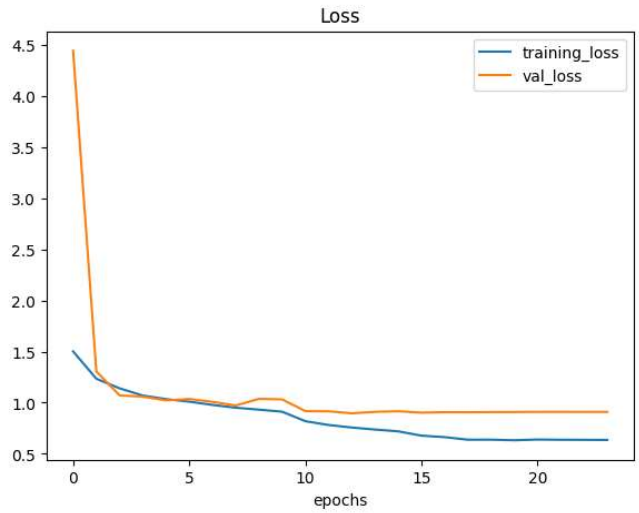
```
448/448 [==============================] - 332s 741ms/step - loss: 0.9509 - accuracy

Epoch 9/30

448/448 [==============================] - 333s 742ms/step - loss: 0.9319 - accuracy

Epoch 10/30

448/448 [==============================] - ETA: 0s - loss: 0.9118 - accuracy: 0.6692

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

448/448 [==============================] - 327s 730ms/step - loss: 0.9118 - accuracy

Epoch 11/30

448/448 [==============================] - 330s 736ms/step - loss: 0.8192 - accuracy

Epoch 12/30

448/448 [==============================] - 328s 732ms/step - loss: 0.7819 - accuracy

Epoch 13/30

448/448 [==============================] - 327s 729ms/step - loss: 0.7570 - accuracy
```

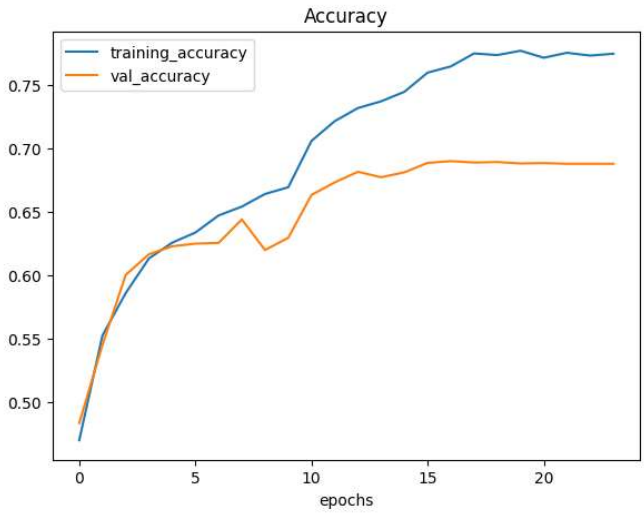
Evaluating ResNet50V2

```
ResNet50V2_Score = ResNet50V2_Model.evaluate(test_data)

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

```
113/113 [==============================] - 20s 176ms/step - loss: 0.9064 - accuracy: 0.6

    Test Loss: 0.90638

Test Accuracy: 69.00%
```

```
plot_curves(ResNet50V2_history)
```



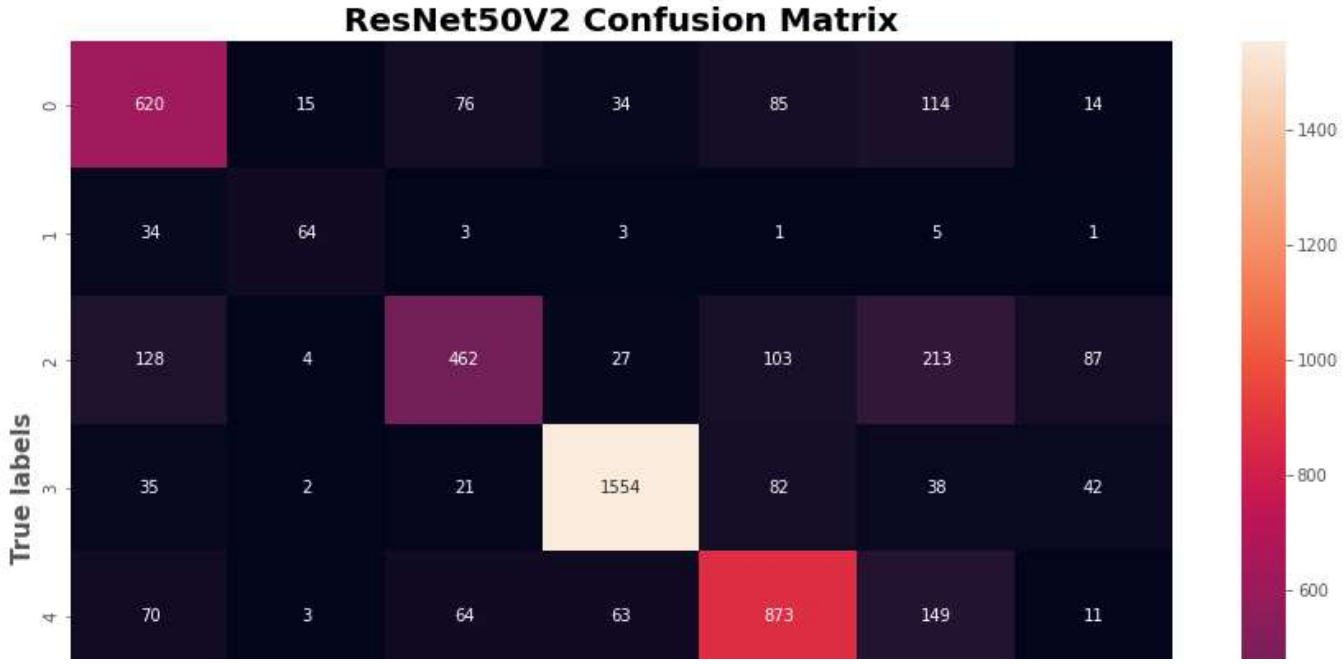



```
ResNet50V2_Predictions = ResNet50V2_Model.predict(test_data)

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

```
     113/113 [==============================] - 19s 163ms/step
```

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

cm=confusion_matrix(test_data.labels, ResNet50V2_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('ResNet50V2 Confusion Matrix', fontsize=20, fontweight='bold')
```

```
Text(0.5, 1.0, 'ResNet50V2 Confusion Matrix')
```



Visualizing Predictions

[ ] ↳ *6 cells hidden*

Music Player

[ ] ↳ *5 cells hidden*

Predicting New Images

Downloading OpenCV haarcascade frontalface Detection

```
!wget https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_f
```

```
faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

```
def load_and_prep_image(filename, img_shape = 224):

    img = cv2.imread(filename)

    GrayImg = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

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

    for x,y,w,h in faces:

        roi_GrayImg = GrayImg[ y: y + h , x: x + w ]
        roi_Img = img[ y: y + h , x: x + w ]

        cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 2)

        plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))

        faces = faceCascade.detectMultiScale(roi_Img, 1.1, 4)

        if len(faces) == 0:
            print("No Faces Detected")
        else:
            for (ex, ey, ew, eh) in faces:
                img = roi_Img[ ey: ey+eh , ex: ex+ew ]

    RGBImg = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

    RGBImg= cv2.resize(RGBImg,(img_shape,img_shape))

    RGBImg = RGBImg/255.

    return RGBImg


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[pred.argmax()]

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

    Recommend_Songs(pred_class)
```

```
pred_and_plot("../input/fer2013/test/sad/PrivateTest_13472479.jpg", Emotion_Classes) # with
```

	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


```
# Downloading Image to Test On
!wget -c "https://pbs.twimg.com/media/EEY3RFFWwAAc-qm.jpg" -O sad.jpg
```

```
pred_and_plot("./happy.jpg", Emotion_Classes) # with CNN
```

	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