emotion_music_recommentation

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Colab AI

[14] 14s

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'fer2013:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F786787%2F1351797%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credent

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading fer2013, 63252113 bytes compressed
[==================================================] 63252113 bytes downloaded
Downloaded and uncompressed: fer2013
Downloading spotify-music-data-to-identify-the-moods, 57310 bytes compressed
[==================================================] 57310 bytes downloaded
Downloaded and uncompressed: spotify-music-data-to-identify-the-moods
Data source import complete.
```

[15] 0s

```
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

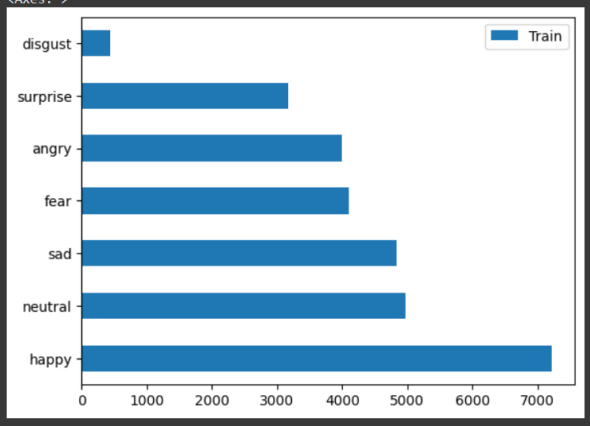
[16]
```
train_dir = '../input/fer2013/train/'
test_dir = '../input/fer2013/test/'

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

    for Class in os.listdir(path):

        Full_Path = 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)
```

[17]
```
pd.concat([Train_Count,Test_Count] , axis=1)
```

	Train	Test
happy	7215	1774
neutral	4965	1233
sad	4830	1247
fear	4097	1024
angry	3995	958
surprise	3171	831
disgust	436	111

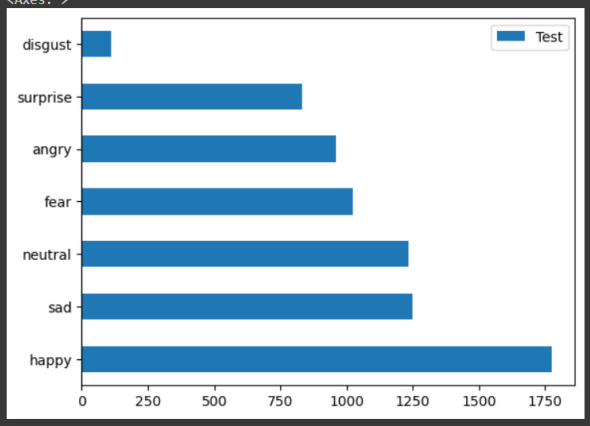
[18]
```
Train_Count.plot(kind='barh')
```

<Axes: >

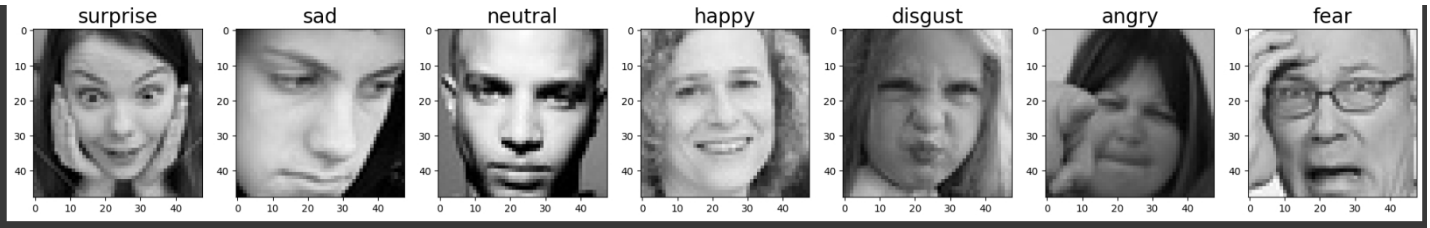


[19]
```
Test_Count.plot(kind='barh')
```

<Axes: >



[20]
```
plt.style.use('default')
plt.figure(figsize = (25, 8))
image_count = 1
BASE_URL = '../input/fer2013/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)
```



Data Preprocessing

```
[21] img_shape = 48
     batch_size = 64
     train_data_path = '../input/fer2013/train/'
     test_data_path = '../input/fer2013/test/'
```

```
[22] 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

```
[23] 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
```

```
[24] CNN_Model = Create_CNN_Model()

     CNN_Model.summary()

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

```
 tchNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 5, 5, 256)         0
 g2D)

 dropout_11 (Dropout)        (None, 5, 5, 256)         0

 flatten_1 (Flatten)         (None, 6400)              0

 dense_7 (Dense)             (None, 1024)              6554624

 batch_normalization_18 (Ba  (None, 1024)              4096
 tchNormalization)

 dropout_12 (Dropout)        (None, 1024)              0

 dense_8 (Dense)             (None, 512)               524800

 batch_normalization_19 (Ba  (None, 512)               2048
 tchNormalization)

 dropout_13 (Dropout)        (None, 512)               0

 dense_9 (Dense)             (None, 256)               131328

 batch_normalization_20 (Ba  (None, 256)               1024
 tchNormalization)

 dropout_14 (Dropout)        (None, 256)               0

 dense_10 (Dense)            (None, 128)               32896

 batch_normalization_21 (Ba  (None, 128)               512
 tchNormalization)

 dropout_15 (Dropout)        (None, 128)               0

 dense_11 (Dense)            (None, 64)                8256

 batch_normalization_22 (Ba  (None, 64)                256
 tchNormalization)

 dropout_16 (Dropout)        (None, 64)                0

 dense_12 (Dense)            (None, 32)                2080

 batch_normalization_23 (Ba  (None, 32)                128
 tchNormalization)

 dropout_17 (Dropout)        (None, 32)                0

 dense_13 (Dense)            (None, 7)                 231

=================================================================
Total params: 7837895 (29.90 MB)
Trainable params: 7832519 (29.88 MB)
Non-trainable params: 5376 (21.00 KB)
_________________________________________________________________
```

Specifying Callbacks

```
[25] # 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 = 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
```

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))
```

```
[ ] 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
```

```
[ ] 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')
```

ResNet50V2 Model

```
[ ] # 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,
)
```

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
```

```
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'])
```

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 = 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,
#                                                    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=30, batch_size=batch_size,
                                          callbacks = callbacks, steps_per_epoch=steps_per_epoch, validation_steps=validation_steps)
```

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))
```

```
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)
```

```
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')
```

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,
    )
```

CNN Predictions

```
[ ] # 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(CNN_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()
```

ResNet50V2 Predictions

```
[ ] # 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()
```

Music Player

```
[ ] Music_Player = pd.read_csv("../input/spotify-music-data-to-identify-the-moods/data_moods.csv")
    Music_Player = Music_Player[['name','artist','mood','popularity']]
    Music_Player.head()
```

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

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

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

```
[ ] # 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'] =='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

Downloading OpenCV haarcascade frontalface Detection

```
[ ] !wget https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml

    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 CNN
```

```
[ ] # 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
```

```
[ ] pred_and_plot("../input/fer2013/test/angry/PrivateTest_22126718.jpg", Emotion_Classes) # with ResNet50V2
```

```
[ ] # Downloading Image to Test On
    !wget -c "https://pbs.twimg.com/profile_images/758370732413947904/xYB5Q3FY_400x400.jpg" -O happy.jpg
```

```
[ ] pred_and_plot("./sad.jpg", Emotion_Classes) # with ResNet50V2
```

```
[ ] CNN_Model.save("CNN_Model.h5")

    ResNet50V2_Model.save("ResNet50V2_Model.h5")
```