Project Name

Live Class Monitoring System(Face Emotion Recognition)

Project Indroduction

The Indian education landscape has been undergoing rapid changes for the past 10 years owing to the advancement of web-based learning services, specifically, eLearning platforms.

Global E-learning is estimated to witness an 8X over the next 5 years to reach USD 2B in 2021. India is expected to grow with a CAGR of 44% crossing the 10M users mark in 2021. Although the market is growing on a rapid scale, there are major challenges associated with digital learning when compared with brick and mortar classrooms.

One of many challenges is how to ensure quality learning for students. Digital platforms might overpower physical classrooms in terms of content quality but when it comes to understanding whether students are able to grasp the content in a live class scenario is yet an open-end challenge.

In a physical classroom during a lecturing teacher can see the faces and assess the emotion of the class and tune their lecture accordingly, whether he is going fast or slow. He can identify students who need special attention.

Digital classrooms are conducted via video telephony software program (exZoom) where it’s not possible for medium scale class (25-50) to see all students and access the mood. Because of this drawback, students are not focusing on content due to lack of surveillance.

While digital platforms have limitations in terms of physical surveillance but it comes with the power of data and machines which can work for you. It provides data in the form of video, audio, and texts which can be analysed using deep learning algorithms.

Deep learning backed system not only solves the surveillance issue, but it also removes the human bias from the system, and all information is no longer in the teacher’s brain rather translated in numbers that can be analysed and tracked.

Method 1 : We use Kaggle for Running our training set

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
# For example, here's several helpful packages to load
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
```

```
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/13288
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/24201
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/26556
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/26076
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/27577
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/27973
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/26452
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/10162
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/12768
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/12551
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/13205
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/21154
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/23053
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/11848
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/22666
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/6797.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/33329
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/7512.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/27926
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/29225
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/5943.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/22019
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/29515
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/35276
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/28482
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/8478.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/28448
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/21074
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/10259
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/32179
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/26800
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/33789
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/35511
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/34664
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/4072.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/19142
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/25687
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/26626
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/35384
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/10800
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/23941
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/6046.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/30336
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/9380.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/4076.
```

```
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/16507
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/19594
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/17693
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/13876
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/9367.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/25229
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/15802
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/34557
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/35823
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/1376.
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/22771
/kaggle/input/face-expression-recognition-dataset/images/validation/surprise/16901
```

Importing Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchN
from keras.models import Model,Sequential
from keras.optimizers import Adam,SGD,RMSprop
```

Displaying Images

```
picture_size = 48
folder_path = "../input/face-expression-recognition-dataset/images/"
```

```
expression = 'disgust'

plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+
                   os.listdir(folder_path + "train/" + expression)[i], target_size=(pictur
    plt.imshow(img)
plt.show()
```

```
batch_size  = 128

datagen_train  = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=True)


test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
```

```
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False)
```

```
Found 28821 images belonging to 7 classes.
Found 7066 images belonging to 7 classes.
```

Making Training and Validation Data

```
from keras.optimizers import Adam,SGD,RMSprop


no_of_classes = 7

model = Sequential()

#1st CNN layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#2nd CNN layer
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#3rd CNN layer
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#4th CNN layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
```

```
# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

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



opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 64)        256
_________________________________________________________________
activation (Activation)      (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 512)       590336
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 512)       2048
_________________________________________________________________
activation_2 (Activation)    (None, 12, 12, 512)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 512)         2359808
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
activation_3 (Activation)    (None, 6, 6, 512)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
```

```
dense (Dense)                (None, 256)               1179904
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
activation_4 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
```

Fitting the Model with Training and Validation Data

```
from keras.optimizers import RMSprop,SGD,Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

checkpoint = ModelCheckpoint("./model.h5", monitor='val_acc', verbose=1, save_best_only=T

early_stopping = EarlyStopping(monitor='val_loss',
                          min_delta=0,
                          patience=3,
                          verbose=1,
                          restore_best_weights=True
                          )

reduce_learningrate = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=3,
                              verbose=1,
                              min_delta=0.0001)

callbacks_list = [early_stopping,checkpoint,reduce_learningrate]

epochs = 48

model.compile(loss='categorical_crossentropy',
              optimizer = Adam(lr=0.001),
              metrics=['accuracy'])
```

```
history = model.fit_generator(generator=train_set,
                              steps_per_epoch=train_set.n//train_set.batch_size,
                              epochs=epochs,
                              validation_data = test_set,
                              validation_steps = test_set.n//test_set.batch_size,
                              callbacks=callbacks_list
                              )
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/training.py:184
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 1/48
225/225 [==============================] - 724s 3s/step - loss: 1.9424 - accuracy: 0.
Epoch 2/48
225/225 [==============================] - 671s 3s/step - loss: 1.4837 - accuracy: 0.
Epoch 3/48
225/225 [==============================] - 671s 3s/step - loss: 1.2894 - accuracy: 0.
Epoch 4/48
225/225 [==============================] - 670s 3s/step - loss: 1.1938 - accuracy: 0.
Epoch 5/48
225/225 [==============================] - 671s 3s/step - loss: 1.1301 - accuracy: 0.
Epoch 6/48
225/225 [==============================] - 671s 3s/step - loss: 1.0757 - accuracy: 0.
Epoch 7/48
225/225 [==============================] - 666s 3s/step - loss: 1.0187 - accuracy: 0.
Epoch 8/48
225/225 [==============================] - 666s 3s/step - loss: 0.9785 - accuracy: 0.
Epoch 9/48
225/225 [==============================] - 668s 3s/step - loss: 0.9380 - accuracy: 0.
Epoch 10/48
225/225 [==============================] - 663s 3s/step - loss: 0.8833 - accuracy: 0.
Epoch 11/48
225/225 [==============================] - 662s 3s/step - loss: 0.8377 - accuracy: 0.
Restoring model weights from the end of the best epoch.

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 00011: early stopping
```

This way we ran epoch it take on my system 3 hrs for only 11 epoch . using early stopping it stop at 11 if we don't use

early stopping it take around 24hrs to ran that epoch because we do not good computational faster computer. We tried most thing like use

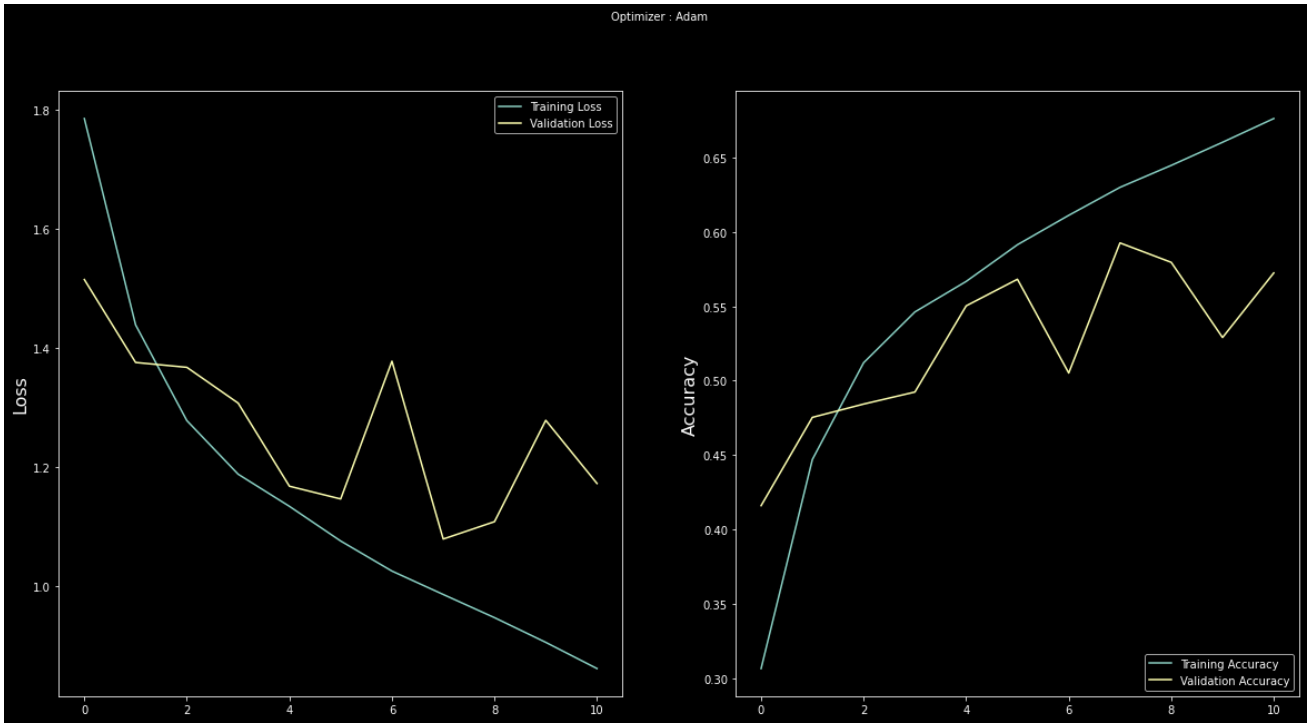
but i approach use transfer learning with new version MobileNet v2 but in my system it not ran properly so that's why we use only two method

Plotting Accuracy & Loss

```
plt.style.use('dark_background')
```

```
plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```



After the all things we plot accuracy graph it quite good accuracy but we want more accuracy because if we use

tranfer learning then accuracy are improve

but we early said we don't have much faster system so this would be best experience for learning purpose and it meet our requirement

We get accuracy in train around 66% and for vaild 56% . But we not stop here we improve the project.

Method 2 : Using deepface we predict the emotion

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install deepface
```

```
Collecting deepface
  Downloading https://files.pythonhosted.org/packages/05/23/f138858cff8750c23a208c
     |████████████████████████████████| 61kB 4.7MB/s
Collecting retina-face>=0.0.1
  Downloading https://files.pythonhosted.org/packages/e7/ce/62209db0f6a9e57120fd36
Requirement already satisfied: Flask>=1.1.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tqdm>=4.30.0 in /usr/local/lib/python3.7/dist-packa
Collecting gdown>=3.10.1
  Downloading https://files.pythonhosted.org/packages/52/b9/d426f164f35bb50d512a77
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Requirement already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: keras>=2.2.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-pack
Collecting mtcnn>=0.1.0
  Downloading https://files.pythonhosted.org/packages/67/43/abee91792797c609c1bf30
     |████████████████████████████████| 2.3MB 17.5MB/s
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: opencv-python>=3.4.4 in /usr/local/lib/python3.7/di
Requirement already satisfied: tensorflow>=1.9.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dis
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dis
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests[socks]>=2.12.0 in /usr/local/lib/python3.7
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages
```

```
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: grpcio~=1.34.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python
Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/local/lib/python3.
Requirement already satisfied: absl-py~=0.10 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: termcolor~=1.1.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0 in /usr/local
Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: typing-extensions~=3.7.4 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/loc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6; extra == "socks" in /usr/lo
Requirement already satisfied: cached-property; python_version < "3.8" in /usr/loc
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/
```

```
import cv2
```

```
img = cv2.imread("/content/drive/MyDrive/face/download.jpg")
```

```
# Analyze Image using DeepFace
from deepface import DeepFace
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f716d5c6310>
```

```
predictions = DeepFace.analyze(img)
```

```
facial_expression_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/facial_expre
To: /root/.deepface/weights/facial_expression_model_weights.h5
100%|██████████| 5.98M/5.98M [00:00<00:00, 17.4MB/s]
age_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/age_model_we
To: /root/.deepface/weights/age_model_weights.h5
100%|██████████| 539M/539M [00:09<00:00, 54.2MB/s]
gender_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/gender_model
To: /root/.deepface/weights/gender_model_weights.h5
100%|██████████| 537M/537M [00:14<00:00, 37.4MB/s]
race_model_single_batch.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/race_model_s
To: /root/.deepface/weights/race_model_single_batch.h5
100%|██████████| 537M/537M [00:07<00:00, 72.3MB/s]
Action: race: 100%|██████████| 4/4 [00:02<00:00,  1.59it/s]
```

```
predictions
```

```
{'age': 39,
 'dominant_emotion': 'angry',
 'dominant_race': 'middle eastern',
 'emotion': {'angry': 99.08357847733241,
  'disgust': 0.006235002381103397,
  'fear': 0.8266986401913929,
  'happy': 6.934323098581078e-06,
  'neutral': 0.004164734624173522,
  'sad': 0.07931691142969267,
  'surprise': 7.756402667743905e-07},
 'gender': 'Man',
 'race': {'asian': 0.6359536200761795,
  'black': 1.041712611913681,
  'indian': 8.63187089562416,
  'latino hispanic': 16.21260643005371,
  'middle eastern': 42.629021406173706,
  'white': 30.848833918571472},
 'region': {'h': 83, 'w': 83, 'x': 92, 'y': 15}}
```

```
predictions["dominant_emotion"]
```

```
'angry'
```

We are trying to draw a rectangle across the face

```
faceCascade = cv2.CascadeClassifier('/content/drive/MyDrive/face/haarcascade_frontalface_
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = faceCascade.detectMultiScale(gray,1.1,4)

for(x,y,w,h) in faces:
  cv2.rectangle(img, (x , y) , (x+w, y+h) , (0 , 255 , 0) , 2)



font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,

            predictions["dominant_emotion"],
            (0, 50),
            font,1,
            (0, 0, 255),
            2,
            cv2.LINE_4) ;


plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f715eeb6750>
```

Thus we have get rectangle on face and predict right emotion after this we create real time emotion detection webcam . Using deepface it quite good but sometime it take wrong like age if you see the age is told 39 that was right as well as worong we try most images in deepface, This was end of our project.

Some real life experience form project

Understand the deep concept of project

Don't afraid to faliure

From more faliure you get more experience and success will come

Never give up

Have some patience good things happen

Try new things and execute your idea

Start coding or generate with AI.