Face Recognition task is accomplished using ***CascadeClassifier*** and ***LBPHFaceRecognizer*** of OpenCV.

Requirements: OpenCV, numpy, pillow and sklearn

"Images" folder contains folders (each person's name) which contains respective images. They are used to label further in our scripts.

Step-1

Images can be given as input in 2 ways:

1. Webcam**:** Run **capture_faces.py** script which requests for person name (whose folder would be created inside Images folder) and image_name (to save the image inside the folder created)
2. Create folder inside Images folder with the person's name and then save images into it.
   Example: If we need to need to cross verify Suresh (vs) Ramesh, create 2 folders Suresh and Ramesh inside Images folder and then upload pics into those folders respectively.

Step-2

Run the script **faces_training.py**

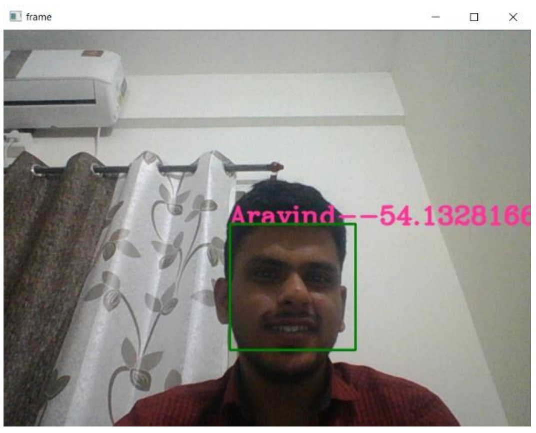
Training the model and saving it. Faces are detected using frontal_cascade of Cascade Classifier and cropped as rectangles using OpenCV. The cropped rectangle is read using PIL and converted into a numpy array. Independent Features(x_train) would be numpy array and Dependent features(y_label) would be the folder name.

model and y_label encoder are saved to the local system.

Step-3

Run the script **face_detection.py**

Model saved earlier is loaded and the labels are decoded using the encoder loaded. Thereby we detect the person and display the person's name along with the confidence percentage.



The recognition task is performed using LBPHFaceRecognizer of OpenCV as training the model using CNN-deep learning keras model takes lot of time and computational power. But however, the script for building the model using keras is below.

```
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import Flatten, Dense, Dropout

from sklearn.metrics import mean_squared_error


x_train = np.reshape(x_train, (550,550,1))

model = Sequential()

model.add(Conv2D(32, kernel_size=(8, 8),activation='relu',input_shape=(550,550,1)))

model.add(MaxPooling2D(pool_size=(2, 2)))
```

.....repeat Conv2D, MaxPooling2D layers

```
model.add(Dropout(0.15))

model.add(Flatten())

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

model.add(Dropout(0.5))

model.add(Dense(len(list(set(y_labels))), activation='softmax'))


model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adadelta(),
metrics=['accuracy'])

model.fit(x_train, y_labels,

        batch_size=72,

        epochs=60,

        verbose=1,

        validation_data=(x_train, y_labels))

score = model.evaluate(x_train, y_labels, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])
```

Model tuning can be done by adjusting the Number of epochs, batch size and number of layers in between and even by dropout

Model Could be saved to local system by:

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
 model.save_weights("model.h5")
```

Model can be loaded into face_detection file using:

```
 json_file = open('model.json', 'r')
 loaded_model_json = json_file.read()
 json_file.close()
 loaded_model = model_from_json(loaded_model_json)
 #...Load weights
 loaded_model.load_weights("model.h5")
```