```
import cv2
import tensorflow as tf
from tensorflow import keras
import os
import sys
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

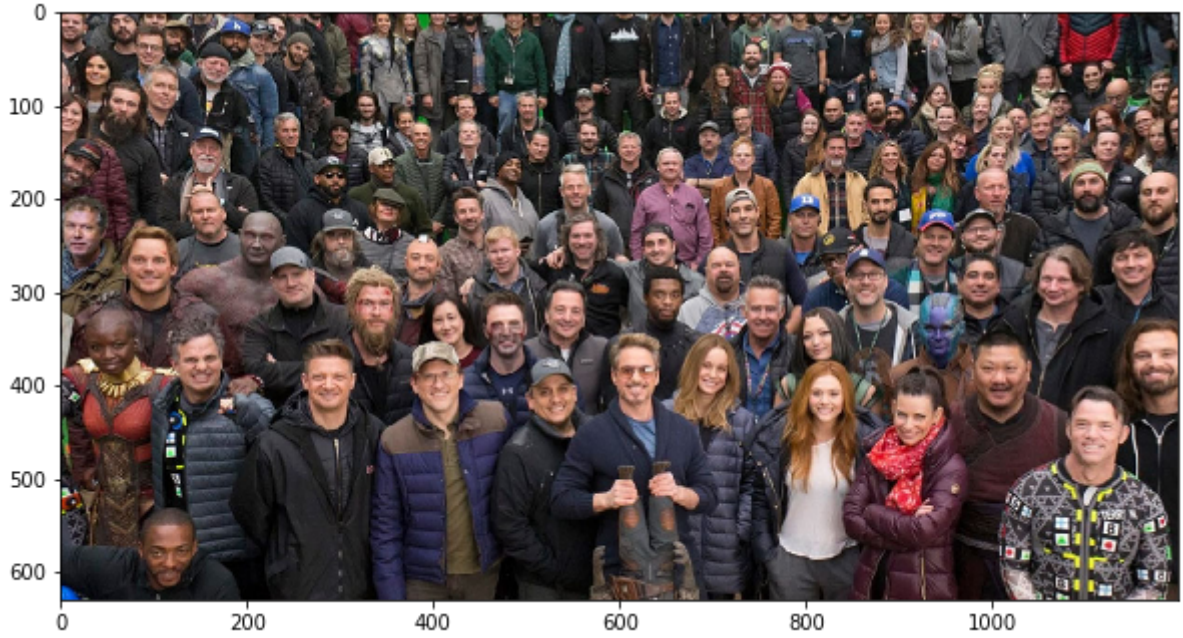
Face Recogniton is implemented :

1. Detect Faces using Viola-Jones Cascade Classifier.
2. Extract the faces from images based on the bounding boxes.
3. Face embeddings will be created using FaceNet which is a model trained by google capable of (1,
4. Using these Face embeddings 2 models will be trained one is an SVM and another is a MLP model.
5. The models will be first trained on the 128x1 vectors and later dimensionality will be reduced
6. Techniques that will be used for dimensionality reduction :
    a. Encoder-Decoder
    b. PCA

```
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

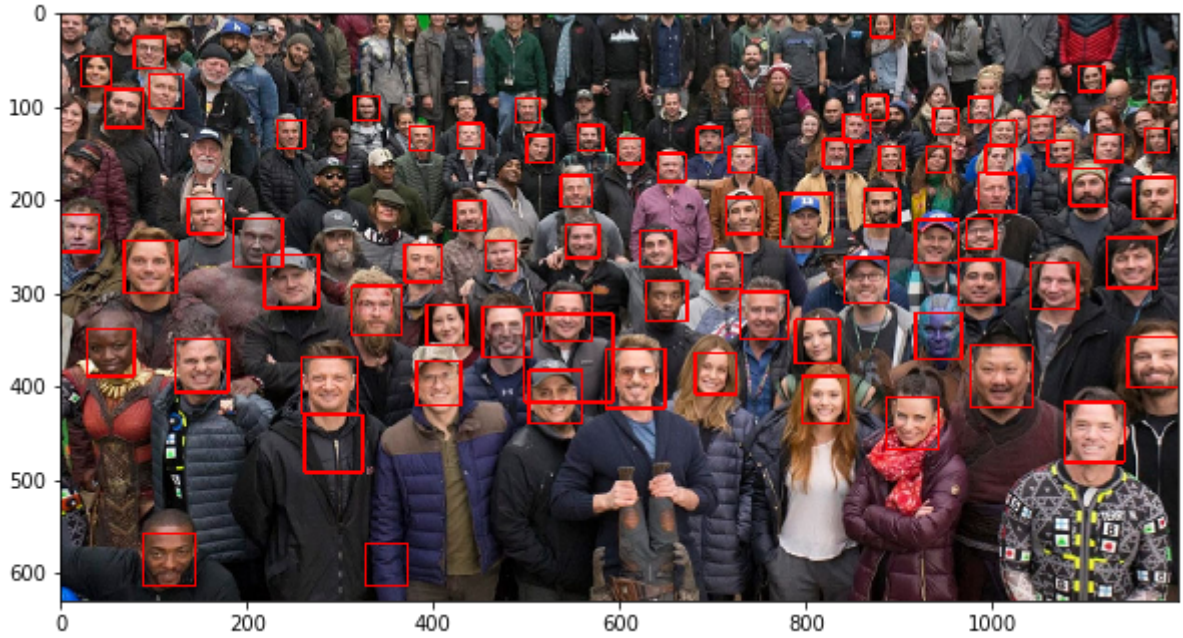
```
img = cv2.imread('test3.jpg')
img2 = cv2.cvtColor(cv2.imread('test3.jpg'),cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x1dd0cfea708>
```



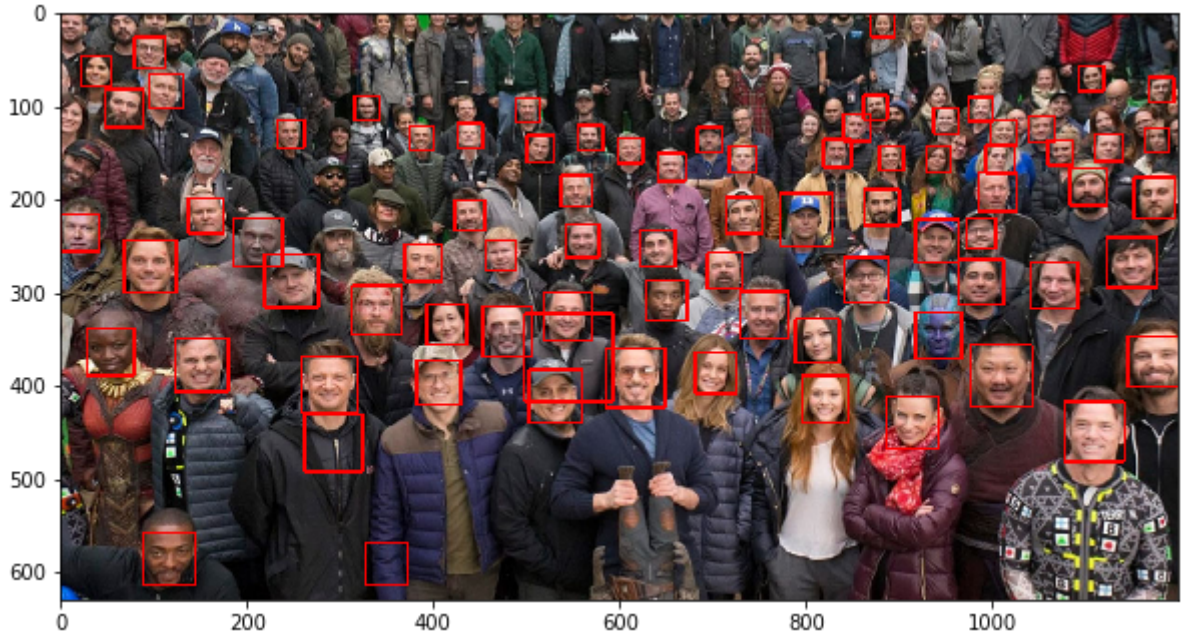
```
bound_boxes = classifier.detectMultiScale(img2,1.05,9)
for x, y, w, h in bound_boxes:
    img2=cv2.rectangle(img2,(x,y),(x+w,y+h),(255,0,0),2)
plt.figure(figsize=(10,10))
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x1dd0d4abec8>
```




```
bound_boxes2 = classifier.detectMultiScale(img2,1.05,9)
for x, y, w, h in bound_boxes:
    img2=cv2.rectangle(img2,(x,y),(x+w,y+h),(255,0,0),2)
plt.figure(figsize=(10,10))
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x1dd0d55af08>
```

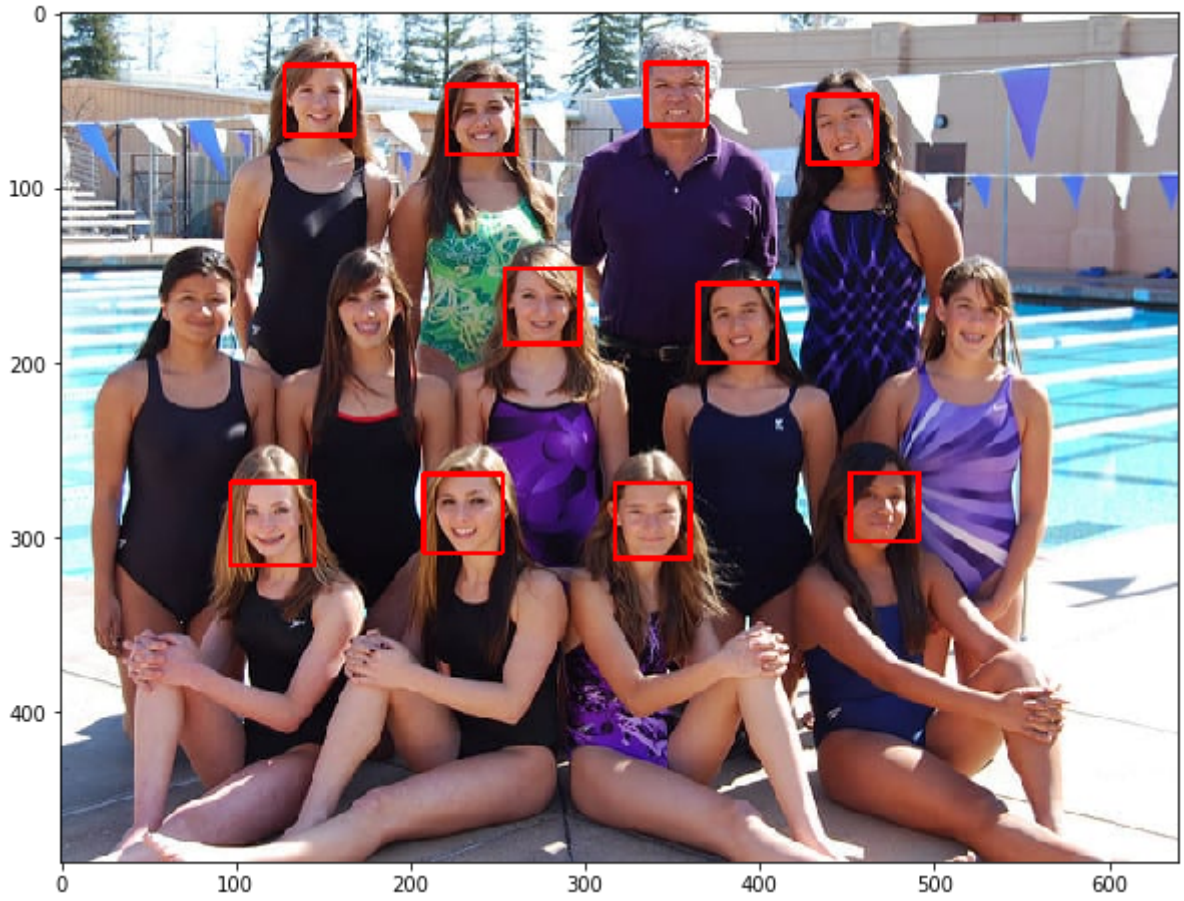


Although the HaarCascade classifier is not giving that good performance it will be used for transforming the traning dataset and later while deployment the HaarCascade model can be

replaced with MTCNN model for Face-Detection.

```
img = cv2.imread('test2.jpg')
img2 = cv2.cvtColor(cv2.imread('test2.jpg'),cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,10))
bound_boxes = classifier.detectMultiScale(img2,1.05,9)
for x, y, w, h in bound_boxes:
    img2=cv2.rectangle(img2,(x,y),(x+w,y+h),(255,0,0),2)
plt.figure(figsize=(10,10))
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x1dd0d5d0c48>
<Figure size 720x720 with 0 Axes>
```




```
# Now extracting the faces from the photograph and showing them.
plt.figure(figsize=(10,10))
def extract_faces(image):
    faces=list()
    i=0
    boxes=classifier.detectMultiScale(img2,1.05,9)
    for x,y,w,h in boxes:
        x2,y2 = x+w,y+h
        face=image[y:y2,x:x2]
        faces.append(face)
        plt.subplot(1,len(boxes),i+1)
        plt.axis('off')
        plt.imshow(face)
        i=i+1
```

```
    return faces
faces=extract_faces(img2)
```

Preparing the Dataset :

1. Extracting cropped face images from dataset reshaping to 160x160 and saving them to another lo
This dimension because FaceNet model Input is of this shape.

```
train_path='./face_images/Final Training Images'
train_act_path='./face_images/training_preped'
test_path='./face_images/Final Testing Images'
test_act_path='./face_images/testing_preped'
faceset = os.listdir(train_path)
for person in faceset:
    images_path = train_path+'/'+person
    images_act_path = train_act_path+'/'+person
    os.mkdir(images_act_path)
    for image in os.listdir(images_path):
        img=cv2.cvtColor(cv2.imread(images_path+'/'+image),cv2.COLOR_BGR2RGB)
        f1=classifier.detectMultiScale(img,1.05,9)
        x,y,w,h=f1[0]
        img=img[y:(y+h),x:(x+w)]
        cv2.imwrite(images_act_path+'/'+image,cv2.cvtColor(img,cv2.COLOR_RGB2BGR))
```

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-8-55f12e7d5682> in <module>
      7     images_path = train_path+'/'+person
      8     images_act_path = train_act_path+'/'+person
----> 9     os.mkdir(images_act_path)
     10     for image in os.listdir(images_path):
     11
img=cv2.cvtColor(cv2.imread(images_path+'/'+image),cv2.COLOR_BGR2RGB)

FileExistsError: [WinError 183] Cannot create a file when that file already exists:
'./face_images/training_preped/face1'
```

```
faceset = os.listdir(test_path)
for person in faceset:
    images_path = test_path+'/'+person
    images_act_path = test_act_path+'/'+person
    os.mkdir(images_act_path)
    for image in os.listdir(images_path):
        img=cv2.cvtColor(cv2.imread(images_path+'/'+image),cv2.COLOR_BGR2RGB)
        f1=classifier.detectMultiScale(img,1.05,9)
        x,y,w,h=f1[0]
        img=img[y:(y+h),x:(x+w)]
        cv2.imwrite(images_act_path+'/'+image,cv2.cvtColor(img,cv2.COLOR_RGB2BGR))
```

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-9-73b42ecf9eaa> in <module>
      3     images_path = test_path+'/'+person
      4     images_act_path = test_act_path+'/'+person
----> 5     os.mkdir(images_act_path)
      6     for image in os.listdir(images_path):
      7
img=cv2.cvtColor(cv2.imread(images_path+'/'+image),cv2.COLOR_BGR2RGB)

FileExistsError: [WinError 183] Cannot create a file when that file already exists:
'./face_images/testing_preped/face1'
```

```
# The dataset is present at train_act_path
# Now this function will only require a folder with sub-folders containing the images o
def load_dataset(folder):
    X, y = list(), list()
    for subdir in os.listdir(folder):
        path = folder +'/'+subdir + '/'
        bb1=os.listdir(path)
        bb1.sort()
        faces = [np.asarray(cv2.resize(cv2.cvtColor(cv2.imread(path+'/'+i),cv2.COLOR_BG
        labels = [subdir for _ in bb1]
        print('>loaded %d examples for class: %s' % (len(faces), subdir))
        X.extend(faces)
        y.extend(labels)
    return np.asarray(X), np.asarray(y)
X,y = load_dataset(train_act_path)
print('-----------------')
X_t,y_t = load_dataset(test_act_path)
```

```
>loaded 15 examples for class: face1
>loaded 20 examples for class: face10
>loaded 16 examples for class: face11
>loaded 14 examples for class: face12
>loaded 13 examples for class: face13
>loaded 12 examples for class: face14
>loaded 15 examples for class: face15
>loaded 17 examples for class: face16
>loaded 15 examples for class: face2
>loaded 14 examples for class: face3
>loaded 17 examples for class: face4
>loaded 16 examples for class: face5
>loaded 16 examples for class: face6
>loaded 14 examples for class: face7
>loaded 14 examples for class: face8
>loaded 16 examples for class: face9
-----------------
>loaded 4 examples for class: face1
>loaded 4 examples for class: face10
>loaded 4 examples for class: face11
>loaded 4 examples for class: face12
>loaded 4 examples for class: face13
>loaded 4 examples for class: face14
>loaded 4 examples for class: face15
>loaded 4 examples for class: face16
```

```
>loaded 4 examples for class: face2
>loaded 4 examples for class: face3
>loaded 4 examples for class: face4
>loaded 4 examples for class: face5
>loaded 4 examples for class: face6
>loaded 4 examples for class: face7
>loaded 4 examples for class: face8
>loaded 4 examples for class: face9
```

```
print(X.shape)
```

```
(244, 160, 160, 3)
```

```
y.shape
```

```
(244,)
```

```
from sklearn.preprocessing import Normalizer
model = keras.models.load_model('facenet_keras.h5')
def get_embed(model,image):
    mean = image.mean()
    std = image.std()
    image=np.array([(image-mean)/std])
    embed=model.predict(image)
    return embed[0]
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
```

```
model.summary()
```

```
Model: "inception_resnet_v1"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
input_1 (InputLayer)            [(None, 160, 160, 3) 0
__________________________________________________________________________________
Conv2d_1a_3x3 (Conv2D)          (None, 79, 79, 32)   864         input_1[0][0]
__________________________________________________________________________________
Conv2d_1a_3x3_BatchNorm (BatchN (None, 79, 79, 32)   96          Conv2d_1a_3x3[0][
__________________________________________________________________________________
Conv2d_1a_3x3_Activation (Activ (None, 79, 79, 32)   0           Conv2d_1a_3x3_Bat
__________________________________________________________________________________
Conv2d_2a_3x3 (Conv2D)          (None, 77, 77, 32)   9216        Conv2d_1a_3x3_Act
__________________________________________________________________________________
Conv2d_2a_3x3_BatchNorm (BatchN (None, 77, 77, 32)   96          Conv2d_2a_3x3[0][
__________________________________________________________________________________
Conv2d_2a_3x3_Activation (Activ (None, 77, 77, 32)   0           Conv2d_2a_3x3_Bat
__________________________________________________________________________________
Conv2d_2b_3x3 (Conv2D)          (None, 77, 77, 64)   18432       Conv2d_2a_3x3_Act
__________________________________________________________________________________
Conv2d_2b_3x3_BatchNorm (BatchN (None, 77, 77, 64)   192         Conv2d_2b_3x3[0][
__________________________________________________________________________________
Conv2d_2b_3x3_Activation (Activ (None, 77, 77, 64)   0           Conv2d_2b_3x3_Bat
__________________________________________________________________________________
```

```
MaxPool_3a_3x3 (MaxPooling2D)   (None, 38, 38, 64)   0          Conv2d_2b_3x3_Act
__________________________________________________________________________________
Conv2d_3b_1x1 (Conv2D)          (None, 38, 38, 80)   5120       MaxPool_3a_3x3[0]
__________________________________________________________________________________
Conv2d_3b_1x1_BatchNorm (BatchN (None, 38, 38, 80)   240        Conv2d_3b_1x1[0][
__________________________________________________________________________________
Conv2d_3b_1x1_Activation (Activ (None, 38, 38, 80)   0          Conv2d_3b_1x1_Bat
__________________________________________________________________________________
Conv2d_4a_3x3 (Conv2D)          (None, 36, 36, 192)  138240     Conv2d_3b_1x1_Act
__________________________________________________________________________________
Conv2d_4a_3x3_BatchNorm (BatchN (None, 36, 36, 192)  576        Conv2d_4a_3x3[0][
__________________________________________________________________________________
Conv2d_4a_3x3_Activation (Activ (None, 36, 36, 192)  0          Conv2d_4a_3x3_Bat
__________________________________________________________________________________
Conv2d_4b_3x3 (Conv2D)          (None, 17, 17, 256)  442368     Conv2d_4a_3x3_Act
__________________________________________________________________________________
Conv2d_4b_3x3_BatchNorm (BatchN (None, 17, 17, 256)  768        Conv2d_4b_3x3[0][
__________________________________________________________________________________
Conv2d_4b_3x3_Activation (Activ (None, 17, 17, 256)  0          Conv2d_4b_3x3_Bat
__________________________________________________________________________________
Block35_1_Branch_2_Conv2d_0a_1x (None, 17, 17, 32)   8192       Conv2d_4b_3x3_Act
__________________________________________________________________________________
Block35_1_Branch_2_Conv2d_0a_1x (None, 17, 17, 32)   96         Block35_1_Branch_
__________________________________________________________________________________
Block35_1_Branch_2_Conv2d_0a_1x (None, 17, 17, 32)   0          Block35_1_Branch_
__________________________________________________________________________________
Block35_1_Branch_1_Conv2d_0a_1x (None, 17, 17, 32)   8192       Conv2d_4b_3x3_Act
__________________________________________________________________________________
Block35_1_Branch_2_Conv2d_0b_3x (None, 17, 17, 32)   9216       Block35_1_Branch_
__________________________________________________________________________________
Block35_1_Branch_1_Conv2d_0a_1x (None, 17, 17, 32)   96         Block35_1_Branch_
__________________________________________________________________________________
Block35_1_Branch_2_Conv2d_0b_3x (None, 17, 17, 32)   96         Block35_1_Branch_
```

```
X_train = []
for image in X:
    embed = get_embed(model,image)
    X_train.append(embed)
X_train=np.asarray(X_train)
```

```
y_train = y.copy()
```

```
X_test = []
for image in X_t:
    embed = get_embed(model,image)
    X_test.append(embed)
X_test=np.asarray(X_test)
```

```
y_test = y_t.copy()
```

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
y_train=enc.fit_transform(y_train)
y_test=enc.transform(y_test)
```

```
y=enc.transform(y)
y_t=enc.transform(y_t)
```

```
#enc1 = Normalizer(norm='l2')
#X_train = enc1.transform(X_train)
#X_test = enc1.transform(X_test)
```

```
enc.classes_
```

```
array(['face1', 'face10', 'face11', 'face12', 'face13', 'face14',
       'face15', 'face16', 'face2', 'face3', 'face4', 'face5', 'face6',
       'face7', 'face8', 'face9'], dtype='<U6')
```

```
from sklearn.svm import SVC
sv_model=SVC(kernel='linear')
sv_model.fit(X_train,y_train)
```

```
SVC(kernel='linear')
```

```
y_pred_train = sv_model.predict(X_train)
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_pred_train,y_train))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        13
           5       1.00      1.00      1.00        12
           6       1.00      1.00      1.00        15
           7       1.00      1.00      1.00        17
           8       1.00      1.00      1.00        15
           9       1.00      1.00      1.00        14
          10       1.00      1.00      1.00        17
          11       1.00      1.00      1.00        16
          12       1.00      1.00      1.00        16
          13       1.00      1.00      1.00        14
          14       1.00      1.00      1.00        14
          15       1.00      1.00      1.00        16

    accuracy                           1.00       244
   macro avg       1.00      1.00      1.00       244
weighted avg       1.00      1.00      1.00       244
```

```
y_pred = sv_model.predict(X_test)
print(classification_report(y_pred,y_test))
print(y_pred)
```

```
print('---')
print(y_test)
```

```
                precision    recall  f1-score   support

             0       1.00      1.00      1.00         4
             1       1.00      1.00      1.00         4
             2       1.00      1.00      1.00         4
             3       1.00      1.00      1.00         4
             4       1.00      1.00      1.00         4
             5       1.00      1.00      1.00         4
             6       1.00      1.00      1.00         4
             7       1.00      1.00      1.00         4
             8       1.00      1.00      1.00         4
             9       1.00      1.00      1.00         4
            10       1.00      1.00      1.00         4
            11       1.00      1.00      1.00         4
            12       1.00      1.00      1.00         4
            13       1.00      1.00      1.00         4
            14       1.00      1.00      1.00         4
            15       1.00      1.00      1.00         4

      accuracy                           1.00        64
     macro avg       1.00      1.00      1.00        64
  weighted avg       1.00      1.00      1.00        64

[ 0  0  0  0  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  5  5  5
  6  6  6  6  7  7  7  7  8  8  8  8  9  9  9  9 10 10 10 10 11 11 11 11
 12 12 12 12 13 13 13 13 14 14 14 14 15 15 15 15]
---
[ 0  0  0  0  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  5  5  5
  6  6  6  6  7  7  7  7  8  8  8  8  9  9  9  9 10 10 10 10 11 11 11 11
 12 12 12 12 13 13 13 13 14 14 14 14 15 15 15 15]
```

```
m=X_t[23].copy()
mn=m.mean()
st=m.std()
m=(m-mn)/st
b1 = model.predict(np.array([m]))
sv_model.predict(b1)
```

```
array([5], dtype=int64)
```

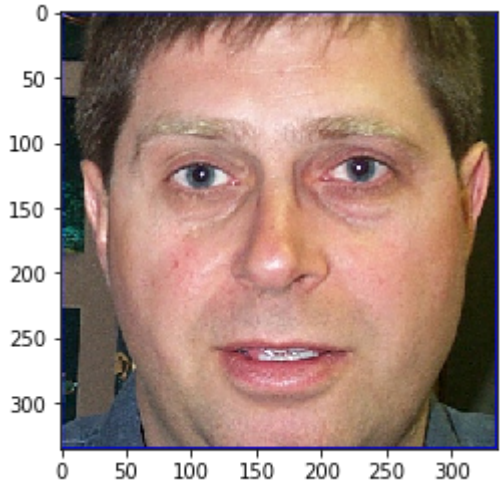
```
print(y_t[23])
```

```
5
```

```
def make_pred(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    f1=classifier.detectMultiScale(img,1.05,9)
    faces=[]
    for x,y,w,h in f1:
        img2=img[y:(y+h),x:(x+w)].copy()
        img2=cv2.resize(img2,(160,160))
        mn=img2.mean()
        st=img2.std()
```

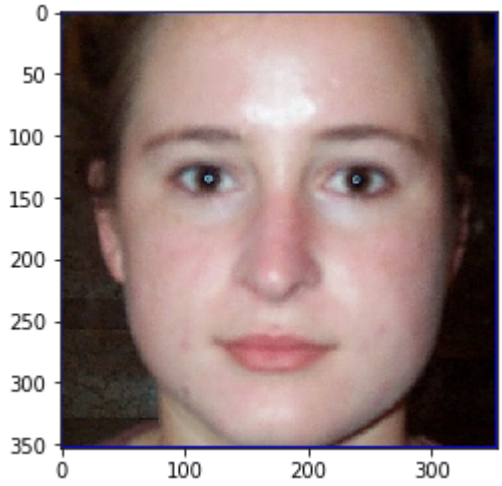
```
        img2=(img2-mn)/st
        emb=model.predict(np.array([img2]))
        pd=sv_model.predict(emb)
        faces.append(pd[0])
    return faces
img=cv2.imread('C:/Users/Jayant Ghadge/Documents/DL_Lab/Assn4/face_images/Final Testing
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
for i in make_pred(img):
    print('This picture contains person number :',i)
```

```
This picture contains person number : 0
```




```
img=cv2.imread('C:/Users/Jayant Ghadge/Documents/DL_Lab/Assn4/face_images/Final Testing
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
for i in make_pred(img):
    print('This picture contains person number :',i)
```

```
This picture contains person number : 13
```




```
1
```

Colab paid products - Cancel contracts here