Face recognition is the general task of identifying and verifying people from photographs of their face. But nowadays and with the high propagation of the **COVID-19**, this task become more and more difficult due to the **masks** weared by the people. In this project, we will discover how to develop a **face recognition system** using **FaceNet** and an SVM and KNN classifiers to identify **Masked People** from photos of people.

Import the reaquired libraries

In [1]:
```
from IPython.display import clear_output
import numpy as np
from numpy import expand_dims
from numpy import asarray

from PIL import Image

from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier

from matplotlib import pyplot
import matplotlib.pyplot as plt
from mtcnn.mtcnn import MTCNN
from os import listdir
```

In [2]:
```
# example of loading the keras facenet model
from keras.models import load_model
```

DataSet

Our data set is composed of two folders: **Train** and **Test**. Each one contains **16 folders of famous people and some friends**. For the train, each person has **15 pictures** in average while for the test we have **5 pictures** in average. We took the **Train Dataset** and we passed it throught an algorithm that takes a **non masked person as an input** and gives us as an output the **same picture but with mask.**

The work is divided into three parts:

- Face Detection
- Face Embedding
- Face Classification

FACE DETECTION

For this task we will use a **deep learning model: MTCNN.**

We have three functions:

- **extract_face**: to extract a face from a given picture
- **load_faces**: to load faces from a given folder
- **load_dataset**: to load the whole dataset

In [3]:
```
# confirm mtcnn was installed correctly
import mtcnn
# print version
print(mtcnn.__version__)
```

```
0.1.0
```

In [4]:
```
# extract a single face from a given photograph
def extract_face(filename, required_size=(160, 160)):
        # load image from file
        image = Image.open(filename)
        # convert to RGB, if needed
        image = image.convert('RGB')
        # convert to array
        pixels = asarray(image)
        # create the detector, using default weights
        detector = MTCNN()
        # detect faces in the image
        results = detector.detect_faces(pixels)
        # extract the bounding box from the first face
        x1, y1, width, height = results[0]['box']
        # bug fix
        x1, y1 = abs(x1), abs(y1)
        x2, y2 = x1 + width, y1 + height
        # extract the face
        face = pixels[y1:y2, x1:x2]
        # resize pixels to the model size
        image = Image.fromarray(face)
        image = image.resize(required_size)
        face_array = asarray(image)
        return face_array
```

In [5]:
```
# load images and extract faces for all images in a directory
def load_faces(directory):
        faces = list()
        # enumerate files
        for filename in listdir(directory):
                # path
                path = directory + filename
                # get face
                face = extract_face(path)
                # store
                faces.append(face)
        return faces
```

In [6]:
```
# load a dataset that contains one subdir for each class that in turn contains images
def load_dataset(directory):
        X, y = list(), list()
        # enumerate folders, on per class
        for subdir in listdir(directory):
                # path
                path = directory + subdir + '/'
                # skip any files that might be in the dir
                #if not isdir(path):
                        #continue
                # load all faces in the subdirectory
                faces = load_faces(path)
                # create labels
                labels = [subdir for _ in range(len(faces))]
                # summarize progress
                print('>loaded %d examples for class: %s' % (len(faces), subdir))
                # store
                X.extend(faces)
                y.extend(labels)
        return asarray(X), asarray(y)
```

In [7]:
```
# load train dataset
trainX, trainy = load_dataset('train/')
print(trainX.shape, trainy.shape)
clear_output()
```

In [8]:
```
# load test dataset
testX, testy = load_dataset('val/')
print(testX.shape, testy.shape)
clear_output()
```

In [9]:
```
# save arrays to one file in compressed format
np.savez_compressed('masked-faces-dataset.npz', trainX, trainy, testX, testy)
```

In [10]:
```
# load the face dataset
data = np.load('masked-faces-dataset.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
print('Loaded: ', trainX.shape, trainy.shape, testX.shape, testy.shape)
```

```
Loaded:  (427, 160, 160, 3) (427,) (105, 160, 160, 3) (105,)
```

As mentionned in the output of the last cell: we have **427 pictures** for the **train** and **105** for the **test**.

Face Embeddings

In this part, We will use **Face Net**:It is a system that, given a picture of a face, will extract high-quality features from the face and predict a 128 element vector representation these features, called a face embedding. These **face embeddings** will then used as the basis for training classifier systems on standard face recognition benchmark datasets

In [11]:
```
# load the model
model = load_model('facenet_keras.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* c
ompiled. Compile it manually.
```

In [12]:
```
# get the face embedding for one face
def get_embedding(model, face_pixels):
        # scale pixel values
        face_pixels = face_pixels.astype('float32')
        # standardize pixel values across channels (global)
        mean, std = face_pixels.mean(), face_pixels.std()
        face_pixels = (face_pixels - mean) / std
        # transform face into one sample
        samples = expand_dims(face_pixels, axis=0)
        # make prediction to get embedding
        yhat = model.predict(samples)
        return yhat[0]
```

In [13]:
```
# convert each face in the train set to an embedding
newTrainX = list()
for face_pixels in trainX:
        embedding = get_embedding(model, face_pixels)
        newTrainX.append(embedding)
newTrainX = asarray(newTrainX)
print(newTrainX.shape)
```

```
(427, 128)
```

In [14]:
```
# convert each face in the test set to an embedding
newTestX = list()
for face_pixels in testX:
        embedding = get_embedding(model, face_pixels)
        newTestX.append(embedding)
newTestX = asarray(newTestX)
print(newTestX.shape)
```

```
(105, 128)
```

In [15]:
```
# save arrays to one file in compressed format
np.savez_compressed('masked-faces-embeddings.npz', newTrainX, trainy, newTestX, testy)
```

In [16]:
```
# load dataset
data = np.load('masked-faces-embeddings.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
print('Dataset: train=%d, test=%d' % (trainX.shape[0], testX.shape[0]))
```

```
Dataset: train=427, test=105
```

In [17]:
```
# normalize input vectors
in_encoder = Normalizer(norm='l2')
trainX = in_encoder.transform(trainX)
testX = in_encoder.transform(testX)
```

In [18]:
```
# label encode targets
out_encoder = LabelEncoder()
out_encoder.fit(trainy)
trainy = out_encoder.transform(trainy)
testy = out_encoder.transform(testy)
```

Face Classification

SVC

In [19]:
```
# fit model
model = svm.SVC(kernel='linear')
model.fit(trainX, trainy)
```

Out[19]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [20]:
```
# predict
yhat_train = model.predict(trainX)
yhat_test = model.predict(testX)
```

Evaluation for the Train

In [21]:
```
print ('Accuracy_train:', accuracy_score(trainy, yhat_train))
print ('F1 score_train:', f1_score(trainy, yhat_train,average='weighted'))
print ('Recall_train:', recall_score(trainy, yhat_train,
                              average='weighted'))
print ('Precision_train:', precision_score(trainy, yhat_train,
                                    average='weighted'))
print ('\n clasification report_train:\n', classification_report(trainy, yhat_train))
print ('\n confussion matrix_train:\n',confusion_matrix(trainy, yhat_train))
```

```
Accuracy_train: 0.990632318501171
F1 score_train: 0.9906293866649055
Recall_train: 0.990632318501171
Precision_train: 0.9913785864100476

 clasification report_train:
               precision    recall  f1-score   support

           0       0.99      1.00      1.00       122
           1       1.00      1.00      1.00       125
           2       1.00      0.90      0.95        10
           3       1.00      0.89      0.94         9
           4       1.00      1.00      1.00        13
           5       0.90      1.00      0.95         9
           6       1.00      1.00      1.00         9
           7       1.00      0.93      0.97        15
           8       1.00      1.00      1.00        17
           9       1.00      1.00      1.00        21
          10       1.00      1.00      1.00        11
          11       1.00      1.00      1.00        15
          12       1.00      0.95      0.98        21
          13       0.89      1.00      0.94        17
          14       1.00      1.00      1.00        13

    accuracy                           0.99       427
   macro avg       0.99      0.98      0.98       427
weighted avg       0.99      0.99      0.99       427


 confussion matrix_train:
 [[122   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 125   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   9   0   0   0   0   0   0   0   0   0   0   1   0]
 [  1   0   0   8   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  13   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   9   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   9   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  14   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0  17   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  21   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  11   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  15   0   0   0]
 [  0   0   0   0   0   1   0   0   0   0   0   0  20   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  17   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  13]]
```

Evaluation for the Test

In [22]:
```
print ('Accuracy_test:', accuracy_score(testy, yhat_test))
print ('F1 score_test:', f1_score(testy, yhat_test,average='weighted'))
print ('Recall_test:', recall_score(testy, yhat_test,
                              average='weighted'))
print ('Precision_test:', precision_score(testy, yhat_test,
                                    average='weighted'))
print ('\n clasification report_test:\n', classification_report(testy, yhat_test))
print ('\n confussion matrix_test:\n',confusion_matrix(testy, yhat_test))
```

```
Accuracy_test: 0.8285714285714286
F1 score_test: 0.8185969586437797
Recall_test: 0.8285714285714286
Precision_test: 0.8651251526251525

 clasification report_test:
               precision    recall  f1-score   support

           0       0.77      1.00      0.87        10
           1       0.94      1.00      0.97        15
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         5
           4       0.83      1.00      0.91         5
           5       1.00      0.40      0.57         5
           6       1.00      1.00      1.00         5
           7       0.58      0.78      0.67         9
           8       0.83      1.00      0.91         5
           9       1.00      1.00      1.00         5
          10       1.00      0.44      0.62         9
          11       1.00      0.71      0.83         7
          12       1.00      1.00      1.00         5
          13       0.50      0.86      0.63         7
          14       0.75      0.38      0.50         8

    accuracy                           0.83       105
   macro avg       0.88      0.84      0.83       105
weighted avg       0.87      0.83      0.82       105


 confussion matrix_test:
 [[10  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 15  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  5  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0  0  0  0  0  0]
 [ 2  0  0  0  0  2  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  5  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  7  1  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0  5  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  5  0  0  0  0  0]
 [ 0  1  0  0  1  0  0  1  0  0  4  0  0  2  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  5  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  5  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  6  0]
 [ 0  0  0  0  0  0  0  3  0  0  0  0  0  2  3]]
```

KNN

In [23]:
```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn.fit(trainX, trainy)
```

Out[23]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [24]:
```
from sklearn.metrics import accuracy_score
# predict
yhat_train1 = knn.predict(trainX)
yhat_test1 = knn.predict(testX)
```

Evaluation for the Train

In [25]:
```
print ('Accuracy_train:', accuracy_score(trainy, yhat_train1))
print ('F1 score_train:', f1_score(trainy, yhat_train,average='weighted'))
print ('Recall_train:', recall_score(trainy, yhat_train1,
                              average='weighted'))
print ('Precision_train:', precision_score(trainy, yhat_train1,
                                    average='weighted'))
print ('\n clasification report_train:\n', classification_report(trainy, yhat_train1))
print ('\n confussion matrix_train:\n',confusion_matrix(trainy, yhat_train1))
```

```
Accuracy_train: 1.0
F1 score_train: 0.9906293866649055
Recall_train: 1.0
Precision_train: 1.0

 clasification report_train:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       122
           1       1.00      1.00      1.00       125
           2       1.00      1.00      1.00        10
           3       1.00      1.00      1.00         9
           4       1.00      1.00      1.00        13
           5       1.00      1.00      1.00         9
           6       1.00      1.00      1.00         9
           7       1.00      1.00      1.00        15
           8       1.00      1.00      1.00        17
           9       1.00      1.00      1.00        21
          10       1.00      1.00      1.00        11
          11       1.00      1.00      1.00        15
          12       1.00      1.00      1.00        21
          13       1.00      1.00      1.00        17
          14       1.00      1.00      1.00        13

    accuracy                           1.00       427
   macro avg       1.00      1.00      1.00       427
weighted avg       1.00      1.00      1.00       427


 confussion matrix_train:
 [[122   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0 125   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0  10   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   9   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  13   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   9   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   9   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  15   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  17   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  21   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  11   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  15   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  21   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  17   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  13]]
```

Evaluation for the Test

In [26]:
```
print ('Accuracy_test:', accuracy_score(testy, yhat_test1))
print ('F1 score_test:', f1_score(testy, yhat_test1,average='weighted'))
print ('Recall_test:', recall_score(testy, yhat_test1,
                              average='weighted'))
print ('Precision_test:', precision_score(testy, yhat_test1,
                                    average='weighted'))
print ('\n clasification report_test:\n', classification_report(testy, yhat_test1))
print ('\n confussion matrix_test:\n',confusion_matrix(testy, yhat_test1))
```

```
Accuracy_test: 0.7904761904761904
F1 score_test: 0.7788577881745583
Recall_test: 0.7904761904761904
Precision_test: 0.8159020873306588

 clasification report_test:
               precision    recall  f1-score   support

           0       0.77      1.00      0.87        10
           1       1.00      1.00      1.00        15
           2       1.00      1.00      1.00         5
           3       0.56      1.00      0.71         5
           4       0.71      1.00      0.83         5
           5       1.00      0.40      0.57         5
           6       1.00      1.00      1.00         5
           7       0.38      0.56      0.45         9
           8       1.00      1.00      1.00         5
           9       1.00      1.00      1.00         5
          10       1.00      0.56      0.71         9
          11       0.86      0.86      0.86         7
          12       0.83      1.00      0.91         5
          13       1.00      0.57      0.73         7
          14       0.25      0.12      0.17         8

    accuracy                           0.79       105
   macro avg       0.82      0.80      0.79       105
weighted avg       0.82      0.79      0.78       105


 confussion matrix_test:
 [[10  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 15  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  5  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  5  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  3  0  2  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  5  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  5  0  0  0  1  1  0  2]
 [ 0  0  0  0  0  0  0  0  5  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  5  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  2  0  0  5  0  0  0  1]
 [ 0  0  0  0  0  0  0  1  0  0  0  6  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  5  0  0]
 [ 2  0  0  1  0  0  0  0  0  0  0  0  0  4  0]
 [ 1  0  0  0  1  0  0  5  0  0  0  0  0  0  1]]
```

Here is a summuary for both the **SVM** and **KNN** classifiers

KNN

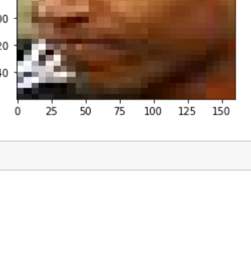
Metrics	Train	Test
Accuracy	1.0	0.79
F1 score	0.99	0.77
Recall	1.0	0.79
Precision	1.0	0.81

SVM

Metrics	Train	Test
Accuracy	0.99	0.82
F1 score	0.99	0.81
Recall	0.99	0.82
Precision	0.99	0.86

In [27]:
```
# develop a classifier for the 5 Celebrity Faces Dataset
from random import choice
from numpy import load
from numpy import expand_dims
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVC
from matplotlib import pyplot
# load faces
data = load('masked-faces-dataset.npz')
testX_faces = data['arr_2']
# load face embeddings
data = load('masked-faces-embeddings.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
# normalize input vectors
in_encoder = Normalizer(norm='l2')
trainX = in_encoder.transform(trainX)
testX = in_encoder.transform(testX)
# label encode targets
out_encoder = LabelEncoder()
out_encoder.fit(trainy)
trainy = out_encoder.transform(trainy)
testy = out_encoder.transform(testy)
# fit model
model = SVC(kernel='linear', probability=True)
model.fit(trainX, trainy)
# test model on a random example from the test dataset
selection = choice([i for i in range(testX.shape[0])])
random_face_pixels = testX_faces[selection]
random_face_emb = testX[selection]
random_face_class = testy[selection]
random_face_name = out_encoder.inverse_transform([random_face_class])
# prediction for the face
samples = expand_dims(random_face_emb, axis=0)
yhat_class = model.predict(samples)
yhat_prob = model.predict_proba(samples)
# get name
class_index = yhat_class[0]
class_probability = yhat_prob[0,class_index] * 100
predict_names = out_encoder.inverse_transform(yhat_class)
print('Predicted: %s (%.3f)' % (predict_names[0], class_probability))
print('Expected: %s' % random_face_name[0])
# plot for fun
pyplot.imshow(random_face_pixels)
title = '%s (%.3f)' % (predict_names[0], class_probability)
pyplot.title(title)
pyplot.show()
```

```
Predicted: dida (61.588)
Expected: dida
```



In [ ]: