

IMAGE PROCESSING

Face Detection and Recognition Project

Intermediate Report

Project ID: 8

PROJECT GROUP MEMBERS:

Alim Kadir KARA - 20190808013

Anıl Can ALPARSLAN - 20190808068

Tahir Emre SEMİZ - 20200808058

Contents

1.INTRODUCTION	2
2.SURVEY	3
3. METHODS.....	4
3.1 Eigenface Method	4
3.1.1 Preprocessing	4
3.1.2 Creating the Eigenfaces.....	5
3.1.3 Recognition.....	7
3.2 Fisherface Method	8
3.2.1 Preprocessing	8
3.2.2 Creating the Training Set.....	8
3.2.3 Creating the Fisherface.....	9
3.2.4 Recognition.....	12
4. DATASET.....	13
5. EXPERIMENTAL RESULTS.....	14
6. SOURCE CODE.....	17
7. REFERENCES	19

1.INTRODUCTION

Face detection and recognition are some of the image processing applications. Recently, the face detection and recognition branch has gained rapid popularity and importance. With this importance, it has developed in many areas both in itself and in the branches it is implemented. Faces are identified and verified in the digital environment with this application. There are multiple methods for this identification process. (It is stated in the continuation of the article.) According to the usage area and features of the work to be done, the advantages and disadvantages of the methods are compared. Thus, the more efficient method is selected and the work begins. By making comparisons between the images obtained in the studies and the dataset, the face recognition process is fully performed.

To perform face recognition, face detection must be done first. After identifying a face in a photo or video, the identity of the face is determined with the dataset. Of course, as in any business, some factors create difficulties in these transactions. Some of these challenges are:

Faces can appear in front of the camera from different angles. These face angles affect the performance of face recognition algorithms.

In poor lighting and high levels of lighting, algorithms cannot work successfully. In the case of instant lighting changes, inconsistencies are experienced in recognition and detection.

Some changes in facial features and accessories such as glasses and hats make it difficult for the algorithm to work properly.

The performance of the algorithms may be disrupted due to the instant facial mimics captured on the camera.

If a face is compared with the dataset years ago, there will be problems in the algorithm. because the algorithm may not catch the obvious changes in the face due to aging.

Some of the areas where face detection and recognition are currently widely used are:

In particular, departing and arriving passengers on international flights are inspected by face recognition systems at airports. Thus, criminals who try to escape or enter the country are arrested and security is ensured.

It is used as a security measure to unlock the phone on smartphones or in applications that require some admin login.

It has gone viral on social media in recent years. It is used to make filters in applications such as Instagram and Snapchat. this technology has made these algorithms effective in the entertainment industry.

2.SURVEY

Some of the commonly used methods are briefly used as follows:

Template Matching: It is not a very successful method for object discrimination. Starting from (1,1) coordinates and moving between all pixels, object detection is made.

Viola Jones algorithm: accuracy rate is high. It is preferred for real-time uses. gives clear results from the front profile. Navigation is done by looking for features related to the face in the picture. (ex. eye, nose, ear, mouth...)

Convolutional Neural Network (CNN): It is a deep learning algorithm. It takes images as input and processes them and classifies them.

Single Shot Detector (SSD): this algorithm performs worse than the CNN algorithm, but is much faster and easier. Because it works fast, it is mostly used for object detection.

Those who will make projects on this subject should investigate these methods in more detail and start their projects by considering their advantages and disadvantages. The above-mentioned face recognition difficulties have been decreasing over the years. With the development of hardware and software, for example, in 2014, the best face recognition algorithm had an error rate of 4.1 percent (ref. 5), but now this rate has decreased to 0.8 percent.

3. METHODS

We will use 2 methods in face recognition. One of them is eigenface and the other is fisherface.

3.1 Eigenface Method

The eigenface method we will use first is a face recognition method created using the Principal Component Analysis (PCA) algorithm. To briefly explain, this algorithm allows to reduce the size in the face space without losing critical information. In this way, when face recognition is performed, it does not need to be navigated in the entire face space, which increases efficiency.

The stages of this method are as follows:

3.1.1 Preprocessing

This step involves how to preprocess the face space. For this purpose, we will make the face space suitable for the PCA algorithm according to 3 normalizations.

- **Size Normalization**

We resize all the images in a common size. (Such as 64x64 pixels or 256x256 pixels) Because in the PCA algorithm, it requires that the data be all in the same dimension.

- **Intensity Normalization**

We normalize the intensity level to grayscale. This step is done because color information is not an important element in eigenface face recognition.

- **Geometric Normalization**

This normalization is critical since there will be many variations (orientation and position of faces) in face space.



Figure 1 (Ref. 1)



Figure 2 (Ref. 1)

3.1.2 Creating the Eigenfaces

After preprocessing, we use PCA to find the "directions" (or eigenvectors) where the data changes the most. In this context, these eigenvectors are actually images, which we call eigenfaces.

- First of all, let's assume that there are m images in $N \times N$ dimensions in our face space.
- We create a matrix where each row corresponds to one face image from the training set. Each image is converted into a one-dimensional vector.

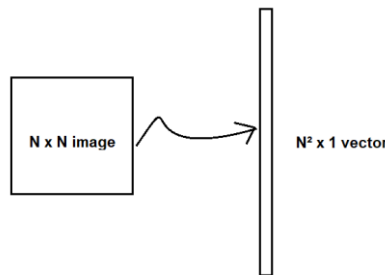


Figure 3 (Ref. 1)

$X_1, X_2, X_3 \dots X_m$

- Then, we calculate the average face of all these face vectors (Figure 5), Then we subtract each face image in the face space from the average face (Figure 6).

$$\psi = \frac{1}{m} \sum_{i=1}^m x_i$$

Figure 6 (Ref. 1)

$$a_i = x_i - \psi$$

Figure 5 (Ref. 1)

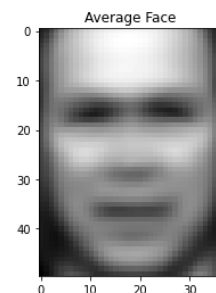


Figure 4 (Ref. 1)

- Then we take all the face vectors and get a matrix in the size of $N^2 \times m$.

$$A = [a_1 \ a_2 \ a_3 \dots a_m]$$

- The covariance matrix of these difference vectors is calculated. This matrix will give a mathematical representation of the set of face images, showing how each pixel in the images relates to every other pixel.

$$\text{Cov} = A^T A$$

- Then, the eigenvectors of the covariance matrix and their corresponding eigenvalues are calculated using the formula (Figure 8).

$$A^T A \nu_i = \lambda_i \nu_i$$

$$A A^T A \nu_i = \lambda_i A \nu_i$$

$$C' u_i = \lambda_i u_i$$

Figure 7

- These eigenvectors, known as eigenfaces in this context, define the features that contribute most to the variation between faces.

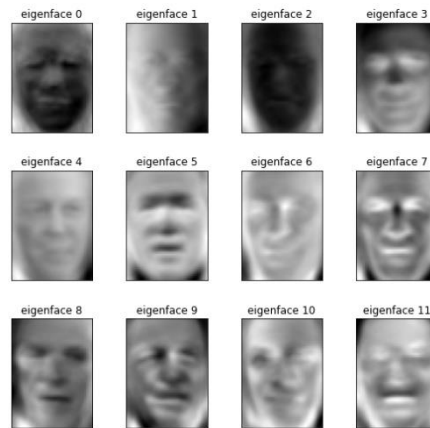


Figure 8 (Ref. 2)

- The chosen eigenfaces are then used to transform the original dataset into a smaller, more manageable size. Each face in the training set is represented as a weighted sum of the chosen eigenfaces.

$$x_i = \begin{bmatrix} w_1^i \\ w_2^i \\ w_3^i \\ \vdots \\ w_k^i \end{bmatrix}$$

=

-2.076 *

-3.046 *

2.127 *

0.637 *

+

+

+

+

-0.758 *

-0.517 *

0.856 *

1.052 *

+

+

+

+

0.458 *

0.013 *

-0.040 *

0.639 *

Figure 9 (Ref. 2)

3.1.3 Recognition

- First of all, we preprocess the image of the unknown face y that we have. Then we subtract the face from the average face.

$$\phi = y - \psi$$

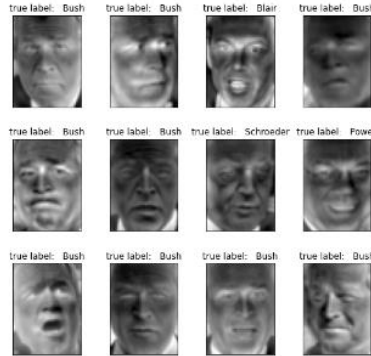


Figure 11 (Ref. 2)

- Then, we generate the vector of the coefficient. We then make him recognize the face by comparing the set of coefficients with those of the training set. We assume that the closest match (using a distance metric) is the correct identity.

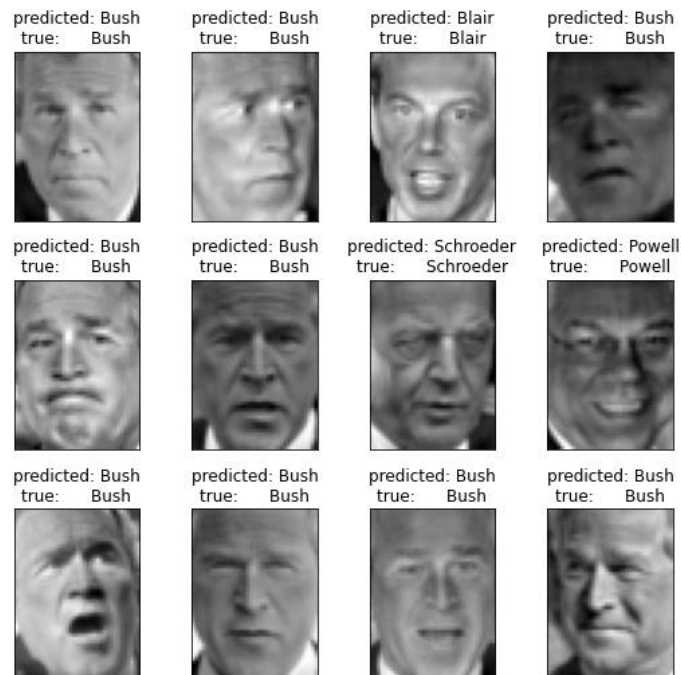


Figure 12 (Ref. 3)

3.2 Fisherface Method

The second fisherface method that we will use is a facial recognition method created using the Linear Discriminatory Analysis (FLDA) technique. To put it more simply, this technique deeply analyzes a lot of different faces and creates a unique identity for each face. And we call it "fisherface". At the same time, one of its most important features is that it can perform robustly even in low-light environments.

The stages of this method are as follows:

3.2.1 Preprocessing

This step involves how to preprocess the face space. For this purpose, we will make the face space suitable for the PCA algorithms according to 3 normalizations.

- **Size Normalization:** We resize all the images in a common size. (Such as 64x64 pixels or 256x256 pixels) Because in the PCA algorithm, it requires that the data be all in the same dimension.
- **Intensity Normalization:** We normalize the intensity level to grayscale. This step is done because color information is not an important element in fisherface recognition.
- **Geometric Normalization:** This normalization is critical since there will be many variations (orientation and position of faces) in face space.

3.2.2 Creating the Training Set

Here is a sample of photos photograph with everyone represented by a minimum of samples of face images with different positions and different expressions.

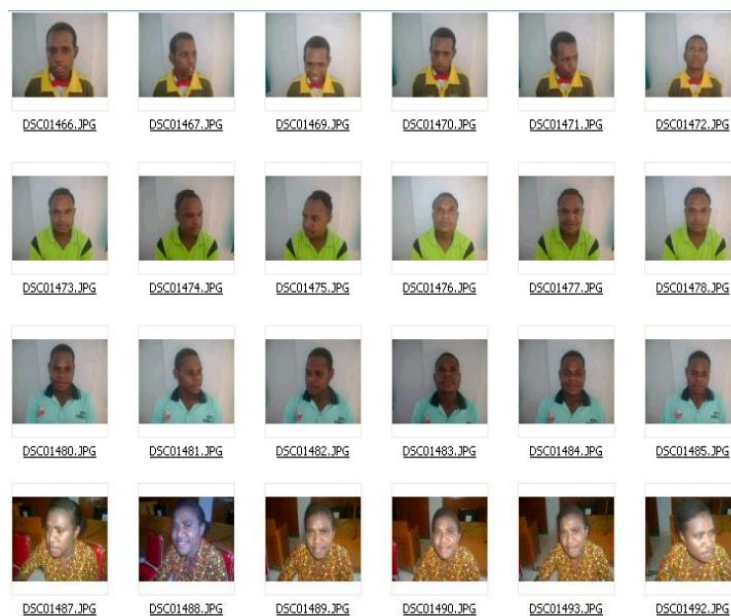


Figure 13 (Ref. 4)



Figure 14. Examples of some training images (Ref. 4)

3.2.3 Creating the Fisherface

Fisherface uses PCA and LDA. And it tries to maximize inter-class scattering and Decrement in-class scattering to a minimum.

3.2.3.1 PCA Algorithm

- Conversion training image $\Gamma_1, 2, \dots, m$ with size $N \times N$ into vector form with length size N^2

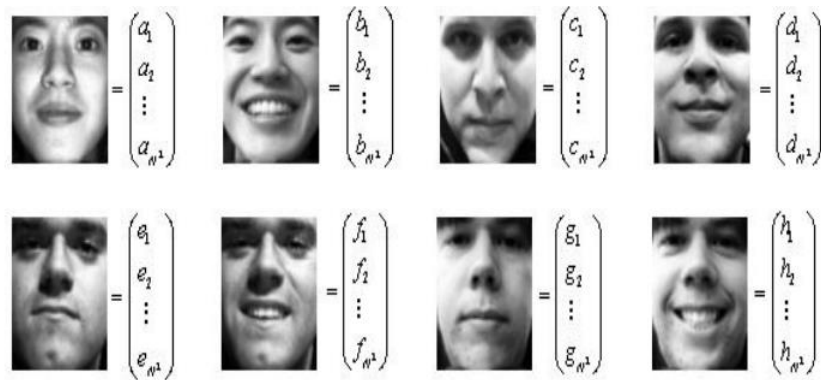


Figure 15 (Ref. 4)

- We determine the mean value across all face images.

$$\vec{m} = \frac{1}{M} \begin{pmatrix} a_1 + b_1 + \dots + h_1 \\ a_2 + b_2 + \dots + h_2 \\ \vdots \\ a_{N^2} + b_{N^2} + \dots + h_{N^2} \end{pmatrix},$$

or written as, $\bar{m} = \frac{\bar{a} + \bar{b} + \dots + \bar{h}}{M}$

- We need to calculate Matrix A. To create this matrix, each image is first converted into a one-dimensional vector by concatenating each row (or each column) of the image. Then, these vectors are combined to form Matrix A. After that we are able to calculate Matrix A.

$$\begin{aligned}\bar{a}_m &= \begin{pmatrix} a_1 - m_1 \\ a_2 - m_2 \\ \vdots \\ a_{N^2} - m_{N^2} \end{pmatrix}, & \bar{b}_m &= \begin{pmatrix} b_1 - m_1 \\ b_2 - m_2 \\ \vdots \\ b_{N^2} - m_{N^2} \end{pmatrix}, \\ \bar{c}_m &= \begin{pmatrix} c_1 - m_1 \\ c_2 - m_2 \\ \vdots \\ c_{N^2} - m_{N^2} \end{pmatrix}, & \bar{d}_m &= \begin{pmatrix} d_1 - m_1 \\ d_2 - m_2 \\ \vdots \\ d_{N^2} - m_{N^2} \end{pmatrix}, \\ \bar{e}_m &= \begin{pmatrix} e_1 - m_1 \\ e_2 - m_2 \\ \vdots \\ e_{N^2} - m_{N^2} \end{pmatrix}, & \bar{f}_m &= \begin{pmatrix} f_1 - m_1 \\ f_2 - m_2 \\ \vdots \\ f_{N^2} - m_{N^2} \end{pmatrix}, \\ \bar{g}_m &= \begin{pmatrix} g_1 - m_1 \\ g_2 - m_2 \\ \vdots \\ g_{N^2} - m_{N^2} \end{pmatrix}, & \bar{h}_m &= \begin{pmatrix} h_1 - m_1 \\ h_2 - m_2 \\ \vdots \\ h_{N^2} - m_{N^2} \end{pmatrix}\end{aligned}$$

$$\bar{A} = [\bar{a} - \bar{m}, \bar{b} - \bar{m}, \dots, \bar{h} - \bar{m}] = [\bar{a}_m, \bar{b}_m, \dots, \bar{h}_m] \text{ or written as,}$$

3.2.3.2 LDA Algorithm

- We calculate the average of each person / class. This allows us to determine the mean feature vectors that set one person or class apart from another.

$$\bar{x} = \frac{1}{2} \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_{N^a} + b_{N^a} \end{pmatrix}, \quad \bar{y} = \frac{1}{2} \begin{pmatrix} c_1 + d_1 \\ c_2 + d_2 \\ \vdots \\ c_{N^a} + d_{N^a} \end{pmatrix},$$

$$\bar{z} = \frac{1}{2} \begin{pmatrix} e_1 + f_1 \\ e_2 + f_2 \\ \vdots \\ e_{N^a} + f_{N^a} \end{pmatrix}, \quad \bar{w} = \frac{1}{2} \begin{pmatrix} g_1 + h_1 \\ g_2 + h_2 \\ \vdots \\ g_{N^a} + h_{N^a} \end{pmatrix}$$

- Then we need construct the scatter matrices S1, S2, S3, S4 in LDA to quantify the spread (or scatter) of the data within each class (the "within-class scatter") and between different classes (the "between-class scatter")

$$S_1 = (\vec{a}_m \vec{a}_m^T + \vec{b}_m \vec{b}_m^T),$$

$$S_2 = (\vec{c}_m \vec{c}_m^T + \vec{d}_m \vec{d}_m^T)$$

$$S_3 = (\vec{e}_m \vec{e}_m^T + \vec{f}_m \vec{f}_m^T),$$

$$S_4 = (\vec{g}_m \vec{g}_m^T + \vec{h}_m \vec{h}_m^T)$$

and matrix within class scatter (ScatW = S1 + S2 + S3 + S4)

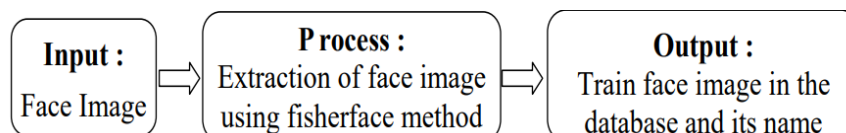
- The construct of also matrix between class scatter, (ScatB)
 $(ScatB = 2(\vec{x} - \vec{m})(\vec{x} - \vec{m})^T + 2(\vec{y} - \vec{m})(\vec{y} - \vec{m})^T + 2(\vec{z} - \vec{m})(\vec{z} - \vec{m})^T + 2(\vec{w} - \vec{m})(\vec{w} - \vec{m})^T)$

So, in here We need to calculate the multiplication of matrices transpose of pe, (PeT), with ScatB and ScatW until we obtain:

$$S_{bb} = PeT * ScatB * Pe$$

$$S_{ww} = PeT * ScatW * Pe$$

Find eigenvector (VeSbb) and generalized eigenvalues (NeSww) of (Sbb, Sww) and we must sort it in such a way that it increases.



Each face is made up of a weighted mixture of Fisherfaces. Projecting the face image into the Fisherface space yields the weights. These weights combine to generate a feature vector that represents the face in Fisherface space.

3.2.4 Recognition

In summary, the Fisherface method works more smoothly and well against changes in lighting and facial expressions. Because it not only reduces dimensionality like PCA, but also maximizes class separability through LDA.























Training	Testing Image	Result
 23910_6_4	 23910_6_1	recognized correctly
 25910_2_3	 25910_2_2	recognized correctly
 25910_3_4	 25910_3_1	recognized correctly
 25910_4_5	 25910_4_2	recognized correctly
 25910_5_4	 25910_3_1	recognized correctly
 23910_4_3	 23910_4_2	recognized correctly
 25910_6_3	 23910_8_1	recognized incorrectly
 25910_7_2	 25910_7_1	recognized correctly
 25910_8_1	 25910_7_1	recognized correctly
 25910_5_4	 25910_9_2	recognized incorrectly
 25910_9_4	 25910_9_1	recognized correctly

Figure 16 (Ref. 4)

4. DATASET

Our dataset consists of a photo collection of 9 different people. Each person is represented by a separate folder.



Each person has a different number of photos. These 9 people have a total of 30 photos.



This dataset was meticulously created to capture the subjects in various positions, angles, and expressions. Images are digitized with sufficient resolution to clearly show key facial features important in recognition tasks such as eyes, nose, mouth, and facial features. Photos are not in grayscale and have just been normalized to a

common size for consistent rendering. These will be converted to grayscale in the PCA algorithm and normalized to a common size.

This dataset provides a broad basis for training our recognition models. The set, which contains more than one photograph of each person, allows the model to understand the variability of the same face in different conditions, while the presence of 9 different people introduces the model to different faces to distinguish it.

5. EXPERIMENTAL RESULTS

For the face recognition task, we have leveraged the Eigenface method, a popular method for face recognition that leverages Principal Component Analysis (PCA) to identify the most distinguishing features across a dataset of faces. To train and evaluate our model, we divided our dataset, which comprises nine folders each containing 30 photos, into training and testing subsets.

Our script reads images from the dataset directory. Each image is converted to grayscale.



After, Each image is resized to a uniform 64x64 size.



The grayscale images are then flattened to a 1-D array and appended to an images list, while their corresponding labels (folder names) are appended to a labels list.

Once the images and labels have been prepared, they are split into a training set (80% of the data) and a testing set (20% of the data). This is achieved by using the `train_test_split` function from scikit-learn.

We then construct a pipeline that includes three steps: standard scaling, PCA, and a support vector machine (SVM) with a radial basis function (RBF) kernel.

StandardScaler is used to standardize the dataset's features to have mean=0 and variance=1, which is a requirement for optimal performance of many machine learning algorithms. PCA is then applied to the standardized images to reduce their dimensionality while retaining 95% of their variance. This transformed data is then classified using the SVM.

After fitting the pipeline with the training data, we use the model to predict the labels of the test data. The model's performance is evaluated by comparing the predicted labels to the true labels and generating a classification report that includes precision, recall, f1-score, and support for each class.

Finally, the code demonstrates how to use the trained model to predict the identity of a single reference photo.



Using the photo above as a reference:

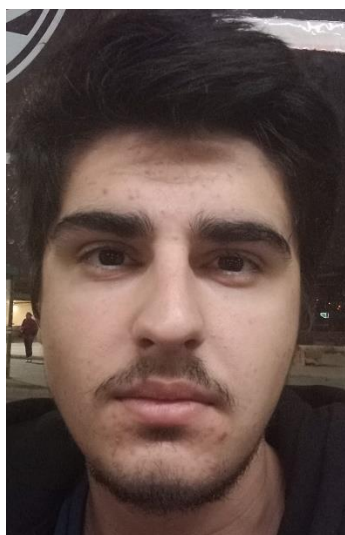
	precision	recall	f1-score	support
anil	0.00	0.00	0.00	0
baran	0.00	0.00	0.00	1
caglar	0.00	0.00	0.00	1
emre	1.00	0.50	0.67	2
enis	0.00	0.00	0.00	0
mete	0.00	0.00	0.00	1
oguz	0.00	0.00	0.00	1
accuracy			0.17	6
macro avg	0.14	0.07	0.10	6
weighted avg	0.33	0.17	0.22	6

Total number of images: 30
The predicted identity of the reference photo is: anil

This experiment has failed. We can talk about this. The given classification report suggests that your machine learning model is not performing well. From the report,

we see that the model's accuracy is just 0.17, meaning it's getting most of the predictions wrong. Here are some potential reasons why the model might be predicting incorrectly:

1. **Insufficient data:** From your report, it seems that you have a very limited amount of data. The total number of images is 30, and some classes (like 'anil' and 'enis') have no test samples. Machine learning models, especially ones based on complex algorithms like Support Vector Machines (SVM), often require larger datasets to generalize effectively.
2. **Imbalanced data:** It seems like some classes might be underrepresented in your dataset. If certain classes have far fewer samples than others, the model might be biased towards predicting the classes with more samples.
3. **Preprocessing and feature extraction:** If the features extracted from the images (or the way they're being preprocessed) are not informative enough, the model might have trouble making accurate predictions.
4. **Model selection or tuning:** The SVM model might not be the best choice for this specific problem, or it might not be correctly tuned. Different types of problems and datasets often require different models or hyperparameters.



When we use the photo above as a reference:

	precision	recall	f1-score	support
anil	0.00	0.00	0.00	0
baran	0.00	0.00	0.00	1
caglar	0.00	0.00	0.00	1
emre	1.00	0.50	0.67	2
enis	0.00	0.00	0.00	0
mete	0.00	0.00	0.00	1
oguz	0.00	0.00	0.00	1
accuracy			0.17	6
macro avg	0.14	0.07	0.10	6
weighted avg	0.33	0.17	0.22	6

Total number of images: 30
The predicted identity of the reference photo is: emre

From this classification report we understand that your model predicts the correct result. As a result, the estimated identity of the reference photo was 'emre' and this is a correct guess. When each of Emre's 6 photographs was used as a reference, it was concluded that 5 of them reached the correct result. This makes the recognition rate of Emre's face 83%.

6. SOURCE CODE

1. The required libraries are imported in the first part:
 - cv2: OpenCV library for image processing
 - numpy: for numerical operations
 - os: to interact with the operating system
 - sklearn: Scikit-learn library for machine learning tasks
 - PCA, StandardScaler, SVC, train_test_split, classification_report: various machine learning tools from Scikit-learn

```
import cv2
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

2. The image dataset directory and the labels (folder names in the directory) are defined. An empty list for images and labels is also created. The target image size is defined as 64x64.

```
dataset_directory = 'dataset'
labels = [f for f in os.listdir(dataset_directory)
          if os.path.isdir(os.path.join(dataset_directory, f))]

images = []
labels_list = []

img_size = (64, 64)
```

3. The script then iterates through each label (folder), reads each image file, resizes it to 64x64, and flattens it (converts the 2D image to a 1D array). It then appends the flattened image array to the images list, and the corresponding label to the labels list.

```
for label in labels:
    path = os.path.join(dataset_directory, label)
    filenames = [f for f in os.listdir(path) if f.endswith('.png') or f.endswith('.jpg')]

    for filename in filenames:
        img = cv2.imread(os.path.join(path, filename), cv2.IMREAD_GRAYSCALE)

        if img is None:
            print(f"Couldn't read file: {filename}")
            continue

        img = cv2.resize(img, img_size)

        img_flattened = img.flatten()
        images.append(img_flattened)

        labels_list.append(label)
```

4. The lists of images and labels are converted to numpy arrays. The dataset is then split into a training set and a test set using the train_test_split function, with 20% of the data reserved for testing.

```
images = np.array(images)
labels = np.array(labels_list)

X_train, X_test, y_train, y_test = train_test_split(images,
                                                    labels,
                                                    test_size=0.2,
                                                    random_state=42)
```

5. A pipeline is created which will standardize the data (using StandardScaler), perform PCA (Principal Component Analysis) to reduce dimensionality (retaining 95% of the variance), and then apply an SVM classifier with a radial basis function (RBF) kernel.

```
# Prepare the pipeline
pipeline = make_pipeline(
    StandardScaler(),
    PCA(n_components=0.95, whiten=True, random_state=42),
    SVC(kernel='rbf')
)
```

6. The SVM model is trained on the training set using the fit function of the pipeline.

```
# Train the model
pipeline.fit(X_train, y_train)
```

7. The trained model is then used to make predictions on the test set, and the performance is evaluated using the `classification_report` function from Scikit-learn, which provides precision, recall, f1-score, and support for each class, as well as overall accuracy.

```
# Make predictions
y_pred = pipeline.predict(X_test)

# Performance evaluation
print(classification_report(y_test, y_pred))
```

8. Finally, the script reads a reference image, preprocesses it in the same way as the other images, and uses the trained model to predict its label (i.e., the category it belongs to).

```
# Prediction of the reference photo
reference_photo_path = 'reference.jpg'

img = cv2.imread(reference_photo_path, cv2.IMREAD_GRAYSCALE)

if img is None:
    print(f"Couldn't read file: {reference_photo_path}")
else:
    img = cv2.resize(img, img_size)
    img_flattened = img.flatten()
    img_array = np.array([img_flattened])
    prediction = pipeline.predict(img_array)
    print(f"Total number of images: {len(images)}")
    print(f"The predicted identity of the reference photo is: {prediction[0]}")
```

7. REFERENCES

1. <https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/>
2. <https://www.atmosera.com/blog/principal-component-analysis/>
3. <https://gamedevacademy.org/face-recognition-with-eigenfaces/>
4. <https://iopscience.iop.org/article/10.1088/1742-6596/1028/1/012119/pdf#:~:text=The%20Design%20System%20Face%20recognition,is%20recognized%20correctly%20or%20not.>
5. <https://www.csis.org/blogs/strategic-technologies-blog/how-accurate-are-facial-recognition-systems-and-why-does-it>