Project Report: Face Recognition with PCA

CelebClone: Find your Celebrity Clone

Yasser Aboussikine

Ikram Ghala

Ismaiiin Boukaidi Laghzaoui

MTH 2303 - 04

EL Mostafa Kalmoun

4/5/2024

Abstract

This project investigates the use of Principal Component Analysis (PCA) for image classification. PCA is a statistical technique that simplifies high-dimensional data by identifying the most important features, known as principal components, that capture the majority of the data's variability. The main goal of this project is to enhance classification efficiency by transforming images into lower-dimensional representations that retain essential characteristics. These principal components are then used to train a classification model, aiming to achieve accurate recognition while minimizing computational complexity. The results demonstrate that PCA effectively preserves the critical information needed for accurate image classification. This report details the methodology, implementation, and outcomes, highlighting the effectiveness of PCA in image analysis tasks.

Table of content

Introduction

This report focuses on using Principal Component Analysis (PCA) for image classification, as implemented in the provided Python code (see appendix). The code applies PCA to reduce the dimensionality of image features extracted using ResNet50, followed by logistic regression for classification. Below, we detail the linear algebra steps of PCA and map them to the code, omitting compression-related components.

Problem Statement or Project Description

2.1 The Problem:

High-level feature vectors extracted from deep neural networks such as ResNet50 can easily exceed 2 000 dimensions. Training a classifier on such high-dimensional data requires more computational resources, lengthens training time, and increases the risk of overfitting. Our challenge is to reduce these feature vectors to a manageable size while preserving the most important information needed for accurate face recognition using PCA.

2.2 Scope and Significance

We decided to work with a controlled subset of the VGGFace2 dataset, limiting ourselves to at most 10 individuals and up to 200 images per person. This setup demonstrates in a clear and reproducible way how PCA can be combined with off-the-shelf deep features. By reducing dimensionality before classification, we achieve faster model fitting, lower memory consumption, and potentially better generalization. This approach is valuable in contexts such as mobile or embedded applications, where both storage and compute capacity are at a premium.

Methodology or Approach

3.1 Overview of Principal Component Analysis (PCA)

PCA reduces data dimensionality by identifying the most significant features (principal components). For classification, PCA helps retain discriminative features while reducing computational complexity.

In simple terms, PCA is a way to find the main “directions” in your data, they're called principal components, those along which the values vary most. Given a data matrix X of size n×d (each of the nnn rows is a ddd-dimensional sample), PCA proceeds as follows:

1- Assemble the data matrix:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix}, \quad x_i \in \mathbb{R}^d$$

2- Compute the mean vector:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

3- Center the data:

$$X_c = X - \mathbf{1}\,\mu^T$$

Where **1** is an n-dimensional vector of ones.

4- For the Covariance matrix:

- The covariance matrix is used to determine the relationships of the features:

$$\Sigma = \frac{1}{n-1}\,X_c^T\,X_c \quad (\text{size } d \times d)$$

5- Eigen decomposition:

- We the compute the eigenvalues of the covariance matrix to find the eigenvectors (the PCs):

$$\Sigma\,v_j = \lambda_j\,v_j \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

- λj are eigenvalues (variance explained by component j).
- vj are eigenvectors (directions of maximum variance).

6- Select top k eigenvectors (principle components):

- We then need to select the top k eigenvectors that hold the most information/align best with the data (the ones that have the most variance).

$$W_k = [v_1, \ldots, v_k] \text{ (}d \times k\text{ matrix)}.$$

7- Project the data:

- The data is projected onto the principal components:

$$Z = X_c W_k \quad (\text{size } n \times k)$$

- Z represents our new matrix with reduced dimensionality where each row is a k-dimensional representation.

-X_c represent the original matrix with centered data.

-Wk is a matrix composed of the top k eigenvectors, with k being the number of principal components chosen.

Implementation or Execution

4.1 Dataset Selection

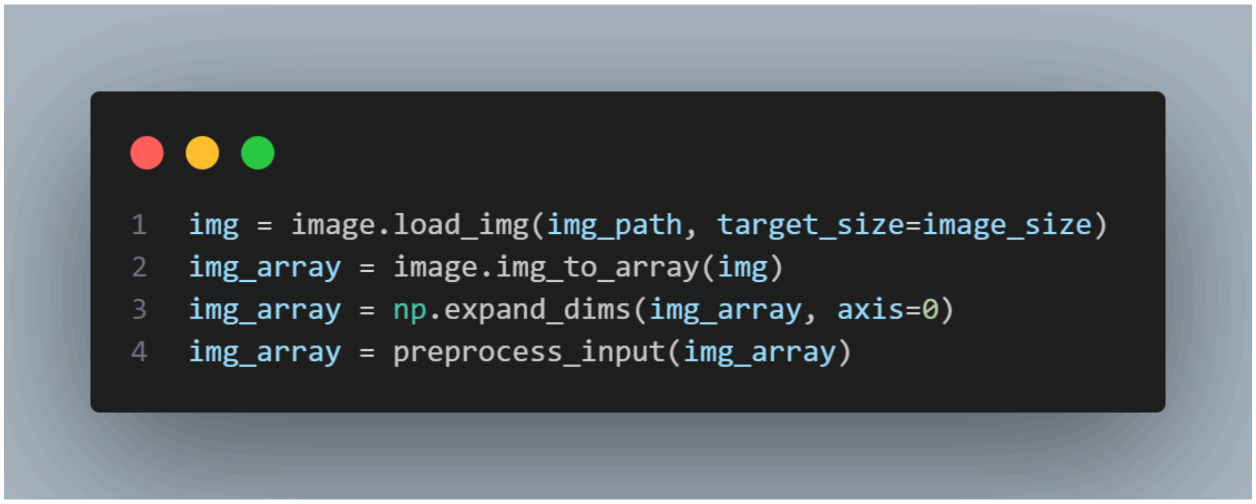
Our choice of the dataset to use wasn't simple, we tried multiple datasets, the first one was the LFW dataset we got by importing " from sklearn.datasets import fetch_lfw_people", but the accuracy was too low, the last option was VGGFace2, which is a very large dataset and it seemed promising, but as it was too large we had to manually remove some classes in order to be able to load it. So our final dataset comprises a subset of the VGGFace2 collection, limited to at most 10 distinct identities with up to 200 images per person.

This selection was made to balance computational feasibility with sufficient intra-class variability. Each image is uniformly resized to 224×224 pixels to satisfy

ResNet50's input requirement. During preprocessing, images are converted from file formats into NumPy arrays and then passed through the preprocess_input function provided by keras.applications.resnet50. This step standardizes pixel intensities to match the distribution on which ResNet50 was originally trained, ensuring that downstream feature extraction remains consistent and robust against variations in lighting and color

4.2 Preprocessing of Images

The preprocessing step loads an image, resizes it to 224x224 pixels, converts it from file formats into NumPy arrays and then passes through the preprocess_input function provided by keras.applications.resnet50. This step standardizes pixel intensities to match the distribution on which ResNet50 was originally trained. This prepares the image for feature extraction using PCA without manual scaling or normalization.

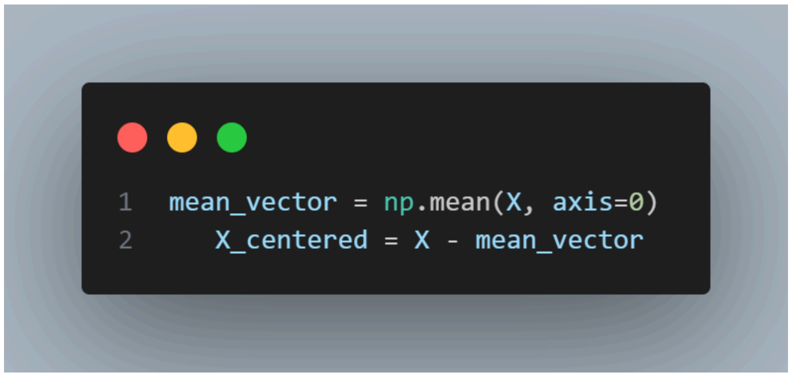


```
img = image.load_img(img_path, target_size=image_size)
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)
```

4.3 PCA Implementation

1- Data Centering

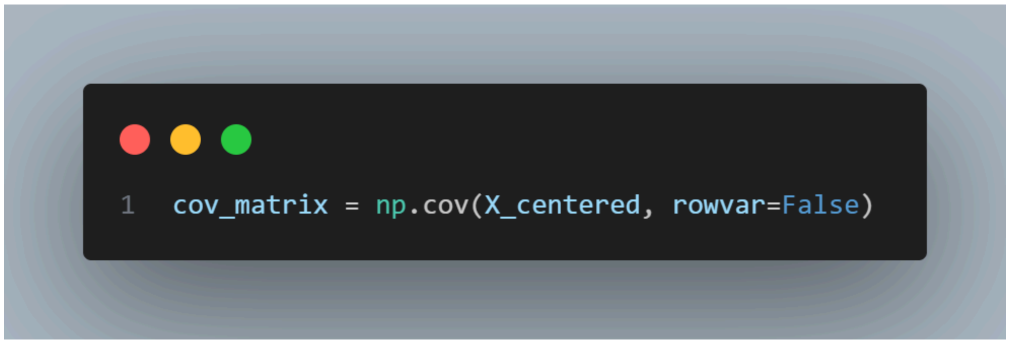
Computes the mean of each feature (column) across all samples. Subtracts the mean from the data, ensuring zero-centered features (critical for PCA's covariance calculation).



```python
mean_vector = np.mean(X, axis=0)
   X_centered = X - mean_vector
```

2- Covariance Matrix

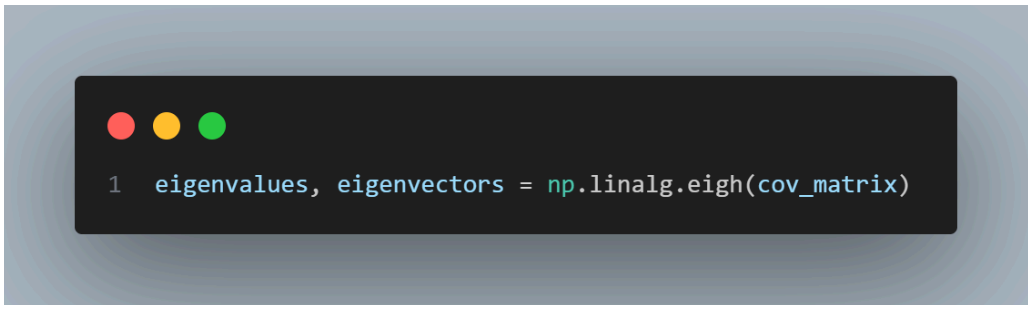
Computes the covariance matrix of the centered data to measure how features vary together.



```python
cov_matrix = np.cov(X_centered, rowvar=False)
```

3- Eigendecomposition

We perform eigendecomposition from the covariance matrix to get the eigenvalues and eigenvectors, the eigenvalues represent the measures of variation of the principal components and the eigenvectors represent the directions of the maximum variance of the data.



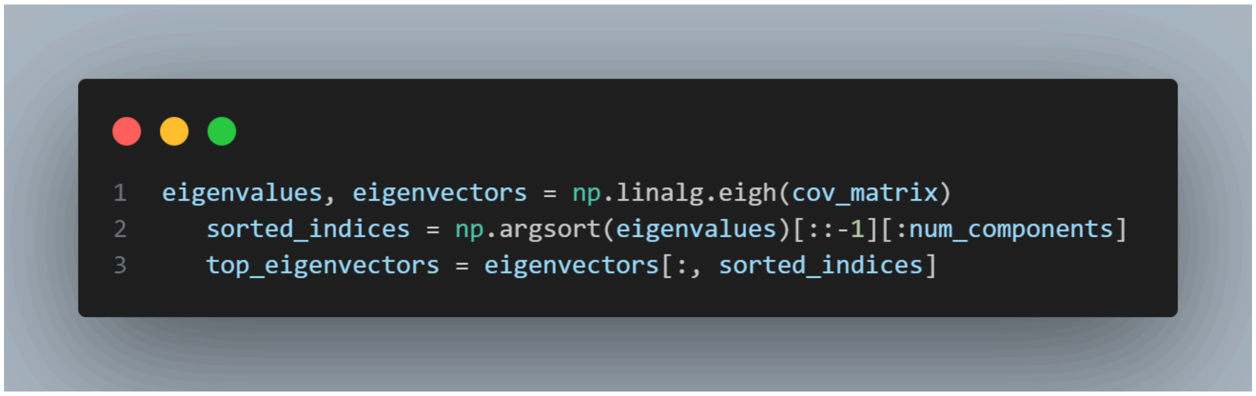
```python
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
```

4- Sorting and Selecting Components

The eigenvalues are sorted from highest variance to lowest and the largest 400 eigenvalues are selected and their associated eigenvectors are extracted.

We chose 400 because it led to the highest accuracy.

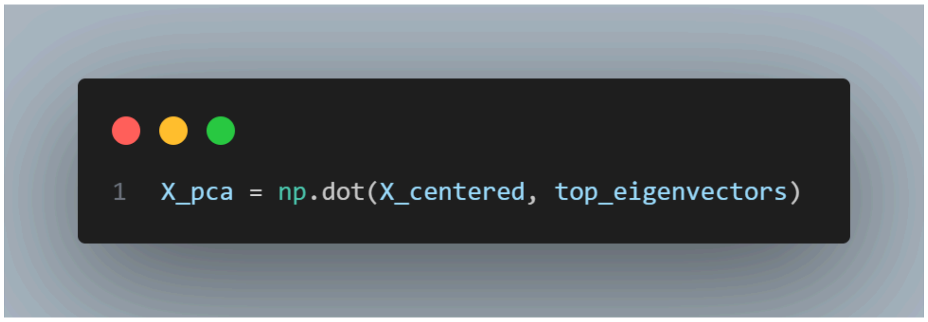
$$Z = R_c W_{400} \quad (n \times 400)$$



```
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    sorted_indices = np.argsort(eigenvalues)[::-1][:num_components]
    top_eigenvectors = eigenvectors[:, sorted_indices]
```

5- Projection to the 400-dim space

Projects the centered data onto the selected PCs.



```
X_pca = np.dot(X_centered, top_eigenvectors)
```

4.4 Image Classification using Logistic Regression

Once the images are compressed using Principal Component Analysis (PCA), we proceed with the classification task using the reduced feature space.

The goal of this step is to identify the person in each image based on the most significant components derived from the original ResNet50 features.

In this project, we used a multinomial Logistic Regression classifier rather than a Support Vector Machine, as we tried it and it failed because the input was too high.

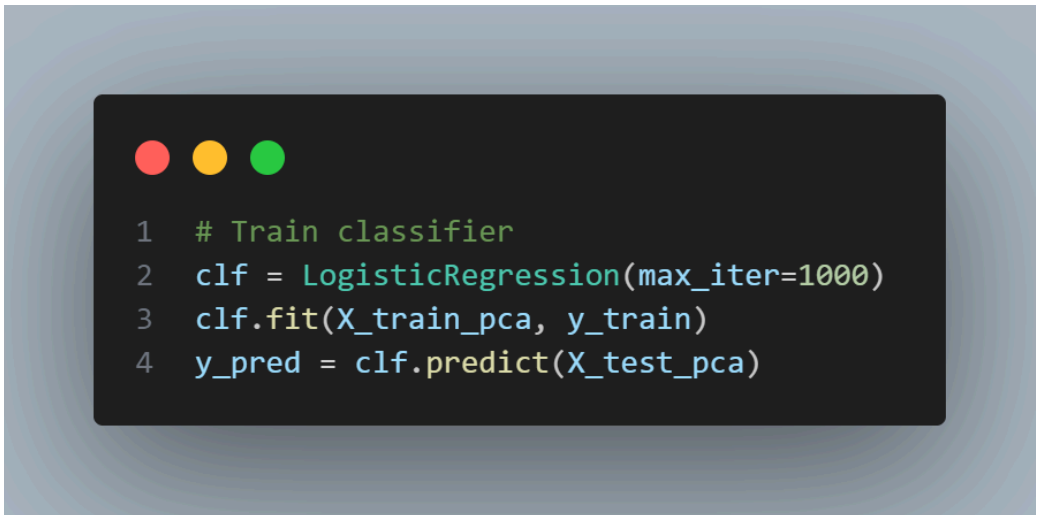
The classifier was trained on the PCA-transformed feature vectors, where each vector represents a face image, and its corresponding label denotes the identity class (each person in the dataset).

After training, the model was evaluated on a separate test set to assess its generalization performance.

Metrics such as accuracy, precision, recall, and F1-score were computed to measure the quality of predictions.

Furthermore, a confusion matrix and classification report were generated to provide a detailed view of the model's performance across different classes.

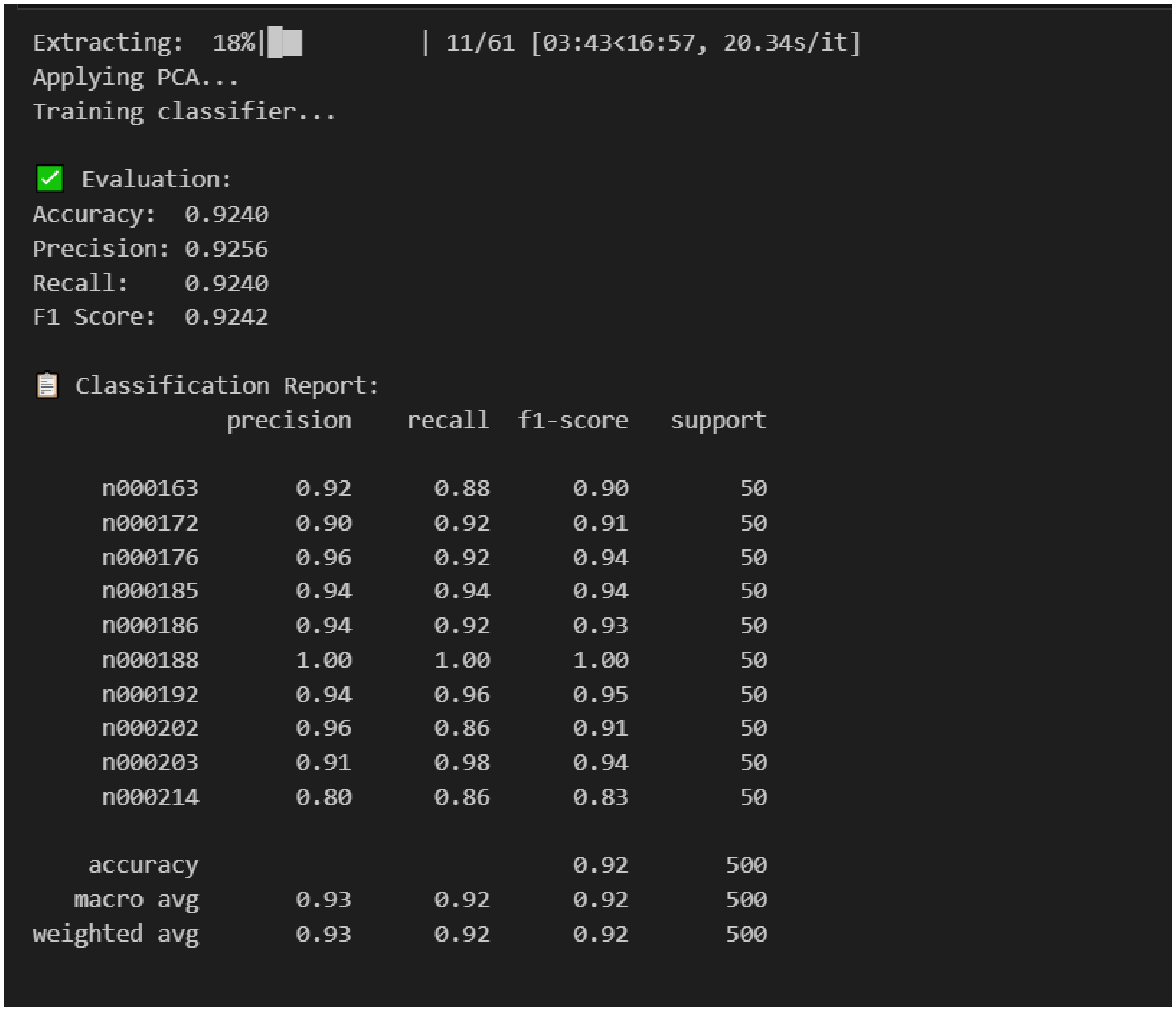
We also visualized predictions through class probability distributions and provided side-by-side comparisons of test images, their PCA representations, and example training images of the predicted class, more details are in the Results section.



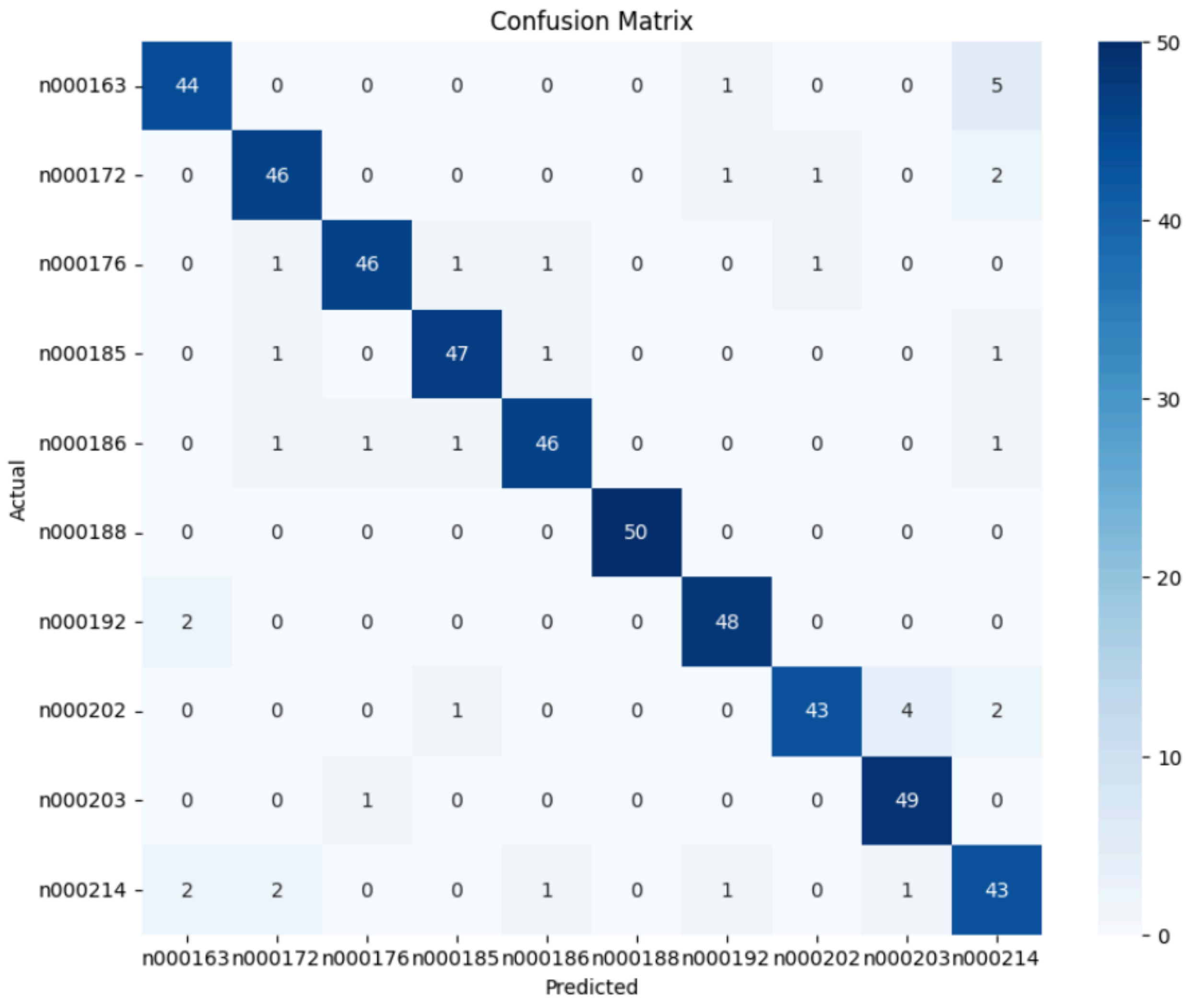
```
# Train classifier
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
```

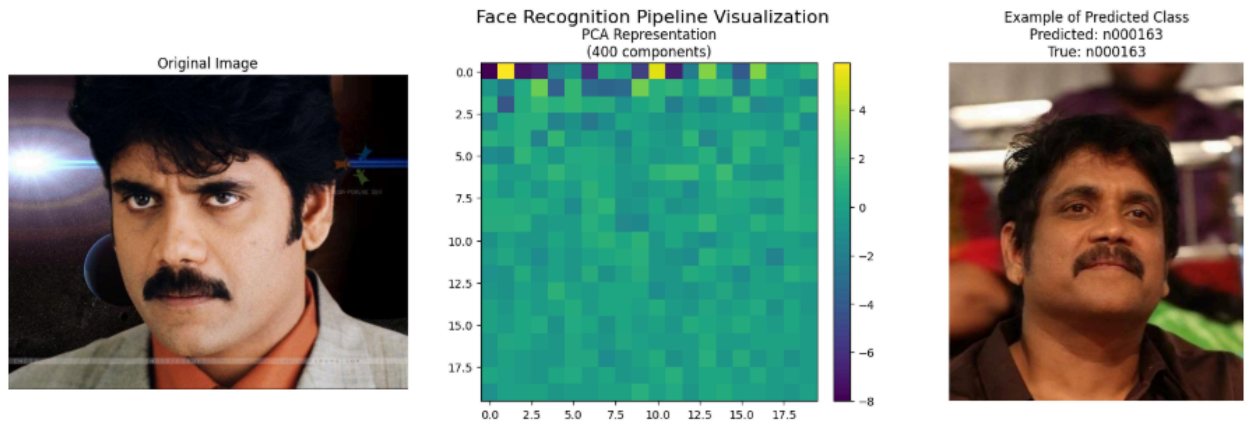
Results and discussion

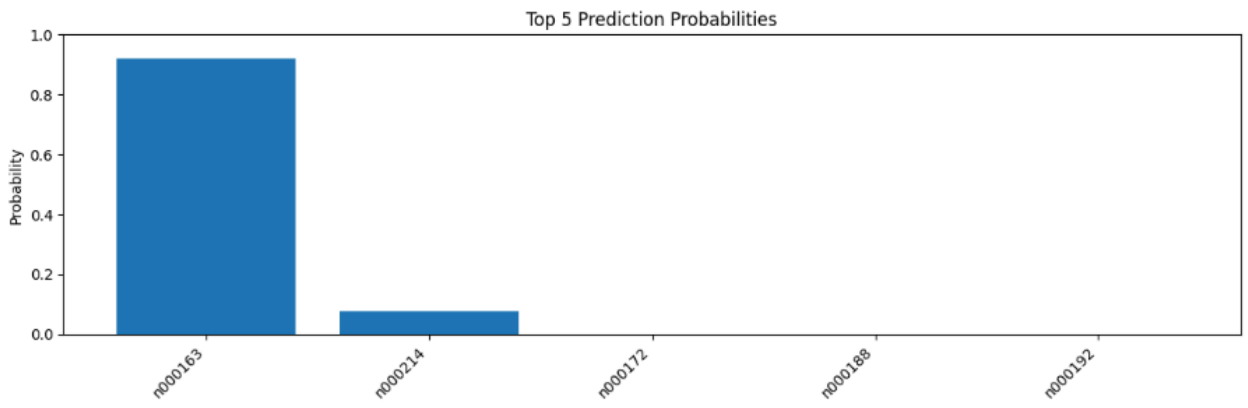
We ended up getting a 92.4% accuracy which is a good result.

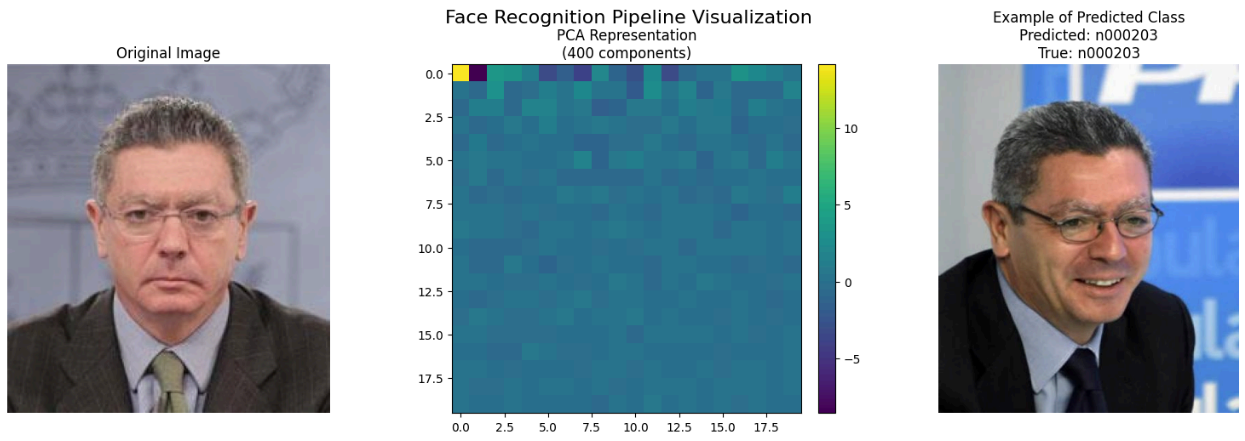


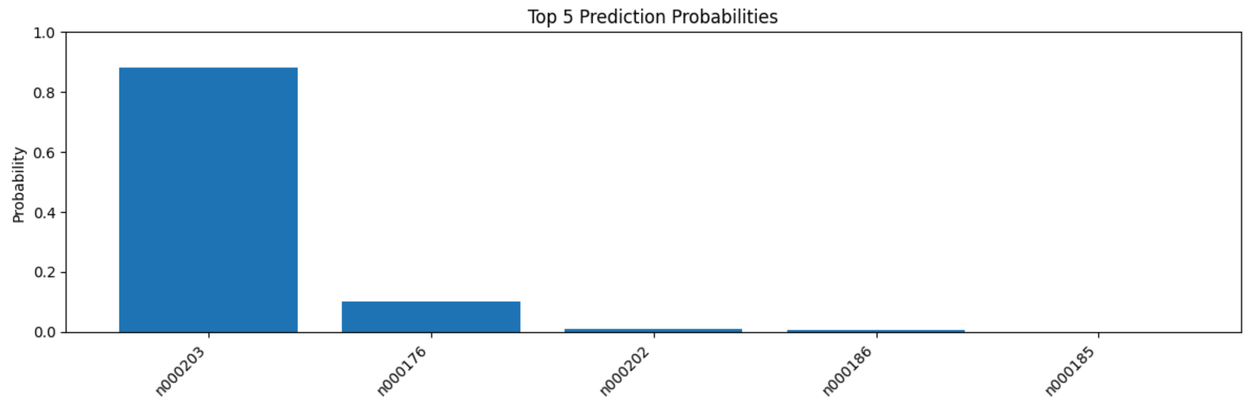
```
Extracting:  18%|██        | 11/61 [03:43<16:57, 20.34s/it]
Applying PCA...
Training classifier...

✅ Evaluation:
Accuracy:  0.9240
Precision: 0.9256
Recall:    0.9240
F1 Score:  0.9242

📋 Classification Report:
              precision    recall  f1-score   support

     n000163       0.92      0.88      0.90        50
     n000172       0.90      0.92      0.91        50
     n000176       0.96      0.92      0.94        50
     n000185       0.94      0.94      0.94        50
     n000186       0.94      0.92      0.93        50
     n000188       1.00      1.00      1.00        50
     n000192       0.94      0.96      0.95        50
     n000202       0.96      0.86      0.91        50
     n000203       0.91      0.98      0.94        50
     n000214       0.80      0.86      0.83        50

    accuracy                           0.92       500
   macro avg       0.93      0.92      0.92       500
weighted avg       0.93      0.92      0.92       500
```


Confusion Matrix
Actual
Predicted
n000163 44 0 0 0 0 0 1 0 0 5
n000172 0 46 0 0 0 0 1 1 0 2
n000176 0 1 46 1 1 0 0 1 0 0
n000185 0 1 0 47 1 0 0 0 0 1
n000186 0 1 1 1 46 0 0 0 0 1
n000188 0 0 0 0 0 50 0 0 0 0
n000192 2 0 0 0 0 0 48 0 0 0
n000202 0 0 0 1 0 0 0 43 4 2
n000203 0 0 1 0 0 0 0 0 49 0
n000214 2 2 0 0 1 0 1 0 1 43
n000163 n000172 n000176 n000185 n000186 n000188 n000192 n000202 n000203 n000214
50
40
30
20
10
0


Face Recognition Pipeline Visualization
Original Image
PCA Representation
(400 components)
Example of Predicted Class
Predicted: n000163
True: n000163


Top 5 Prediction Probabilities
Probability
1.0
0.8
0.6
0.4
0.2
0.0
n000163
n000214
n000172
n000188
n000192


Face Recognition Pipeline Visualization
Original Image
PCA Representation
(400 components)
Example of Predicted Class
Predicted: n000203
True: n000203


Top 5 Prediction Probabilities
Probability
1.0
0.8
0.6
0.4
0.2
0.0
n000203
n000176
n000202
n000186
n000185

References

1. *Principal Component Analysis (PCA) Explained*. Built In,

   https://builtin.com/data-science/covariance-matrix#:~:text=For%20example%2C%20the%20eigenvalues%20represent,the%20covariance%20is%20near%20zero.

2. StatQuest with Josh Starmer. "Principal Component Analysis (PCA), Step-by-Step." *YouTube*, uploaded by StatQuest, 15 Oct. 2015, www.youtube.com/watch?v=3aUshxvxGhY&t=197s.

3. 3Blue1Brown. "Eigenvectors and Eigenvalues Explained Visually." *YouTube*, uploaded by 3Blue1Brown, 14 Mar. 2018, www.youtube.com/watch?v=XQIpnoiow3o&t=667s.