

Final Project Report – Face Recognition using PCA and ANN

1. Introduction

1.1 Project Overview

This project focuses on designing a face recognition system that leverages Principal Component Analysis (PCA) for extracting significant features, also known as eigenfaces, and an Artificial Neural Network (ANN) as the classifier. The workflow begins by converting raw facial images into grayscale, resizing them, and then flattening them into vectors. PCA reduces the dimensionality of these vectors, and the ANN predicts the identity of unseen test faces.

1.2 Objectives

- Apply PCA to decrease dimensionality and capture the most informative features.
- Train an ANN classifier on the PCA-projected feature space.
- Compare recognition performance for different PCA component counts (k values).
- Incorporate a rejection mechanism for imposters based on prediction confidence.

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

Recognizing faces directly from raw pixel intensities is inefficient due to the very high dimensionality of image data. By applying PCA, we can represent each face using a smaller set of principal components. An ANN is then trained to classify these features, producing a practical system for automated face recognition.

2.2 Proposed Solution

The solution includes building a database of face images, applying PCA to compute eigenfaces, and using an ANN classifier. Evaluation is carried out by varying the number of PCA components and measuring accuracy. An imposter detection step is added to ensure that the system can handle unknown individuals.

2.3 Initial Project Planning

Milestone	Deliverable	Timeline
M1	Data preparation and preprocessing	1 day

M2	PCA + ANN baseline model	1 day
M3	Vary k and add imposter detection	2 days
M4	Final documentation and report	2 days

3. Data Collection and Pre-processing Phase

3.1 Data Collection

The dataset is organized into folders where each folder corresponds to a single individual. Each image is a portrait of that person. This structure allows easy mapping between images and labels.

[+ Code](#)[+ Text](#)


```
[14] # =====
# Setup & Imports
# =====
import os, cv2, zipfile
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
```

✓
Os

```
# Step 1: Unzip Dataset
# =====
zip_path = "/content/dataset.zip" # <- Upload dataset.zip to Colab
extract_path = "/content/dataset"

if not os.path.exists(extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)

print("Dataset extracted to:", extract_path)
```

 Dataset extracted to: /content/dataset✓
s

```
# Path to face dataset folder (inside extracted zip)
dir_name = os.path.join(extract_path, "dataset", "faces")
if not os.path.exists(dir_name):
    raise FileNotFoundError(f"Expected folder {dir_name} not found. Please check zip structure.")
```

✓
s

```
[17] # =====
# Utilities
# =====
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(min(n_row * n_col, len(images))):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=9)
        plt.xticks(())
        plt.yticks(())
    plt.show()
```

3.2 Data Pre-processing

Images are converted to grayscale, resized to 300×300 pixels, and then flattened into vectors of length 90,000. This standardized format ensures consistency before applying PCA.

✓
1s



```
# =====  
# Step 2: Load dataset  
# =====  
y, x, target_names = [], [], []  
person_id = 0  
h = w = 300  
n_samples = 0  
class_names = []  
  
for person_name in sorted(os.listdir(dir_name)):  
    dir_path = os.path.join(dir_name, person_name)  
    if not os.path.isdir(dir_path):  
        continue  
    class_names.append(person_name)  
    for image_name in os.listdir(dir_path):  
        image_path = os.path.join(dir_path, image_name)  
        img = cv2.imread(image_path)  
        if img is None:  
            continue  
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
        resized_image = cv2.resize(gray, (h, w))  
        v = resized_image.flatten()  
        x.append(v)
```

```
✓ 1s ▶ resized_image = cv2.resize(gray, (h, w))
v = resized_image.flatten()
x.append(v)
y.append(person_id)
target_names.append(person_name)
n_samples += 1
person_id += 1

x = np.array(x)
y = np.array(y)
n_features = x.shape[1]
n_classes = len(class_names)

print("Total dataset size:")
print("n_samples:", n_samples)
print("n_features:", n_features)
print("n_classes:", n_classes)
```

⇒ Total dataset size:
n_samples: 450
n_features: 90000
n_classes: 9

```
✓ 0s ▶ # Step 3: Train/test split
# =====
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=42, stratify=y
)
```

4. Model Development Phase

4.1 PCA (Eigenfaces)

PCA was applied on the mean-centered face dataset. The eigenvectors of the covariance matrix correspond to the principal directions, also known as eigenfaces. Images are projected onto these directions to obtain compact feature representations.

```

# Step 4: PCA (Eigenfaces)
# =====
n_components = 150
print(f"Extracting the top {n_components} eigenfaces from {X_train.shape[0]} faces")

pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True).fit(X_train)
eigenfaces = pca.components_.reshape((n_components, h, w))

eigenface_titles = [f"eigenface {i}" for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)
plt.show()

print("Projecting the input data on the eigenfaces orthonormal basis")
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("Train PCA shape:", X_train_pca.shape, "Test PCA shape:", X_test_pca.shape)

```

4.2 Classifier (ANN)

A Multi-Layer Perceptron (MLP) was employed with two hidden layers of 10 neurons each. The network was trained using backpropagation, with a maximum of 1000 iterations. The classifier learns to associate eigenface-based features with their respective identities.

✓
Os

```
[21] # =====  
# Step 5: LDA (Fisherfaces)  
# =====  
lda = LinearDiscriminantAnalysis()  
lda.fit(X_train_pca, y_train)  
X_train_lda = lda.transform(X_train_pca)  
X_test_lda = lda.transform(X_test_pca)  
print("LDA projection done...")
```



LDA projection done...

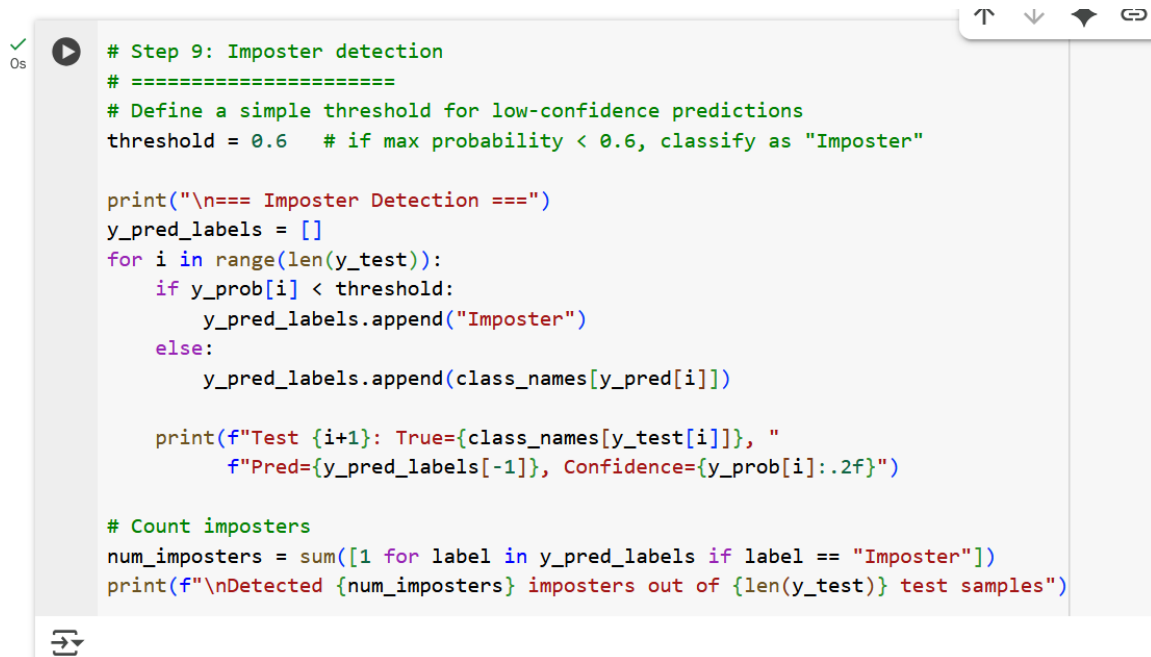
✓
Os



```
# =====  
# Step 6: Classifier (MLP)  
# =====  
clf = MLPClassifier(  
    random_state=1,  
    hidden_layer_sizes=(10, 10),  
    max_iter=1000,  
    verbose=True  
)  
.fit(X_train_lda, y_train)  
  
print("Model weights:", [coef.shape for coef in clf.coefs_])
```

4.3 Imposter Detection

The imposter detection mechanism evaluates the model's confidence. If the maximum probability output is below a chosen threshold (e.g., 0.6), the system classifies the image as an imposter rather than forcing it into a known category.



```
# Step 9: Imposter detection
# =====
# Define a simple threshold for low-confidence predictions
threshold = 0.6 # if max probability < 0.6, classify as "Imposter"

print("\n=== Imposter Detection ===")
y_pred_labels = []
for i in range(len(y_test)):
    if y_prob[i] < threshold:
        y_pred_labels.append("Imposter")
    else:
        y_pred_labels.append(class_names[y_pred[i]])

    print(f"Test {i+1}: True={class_names[y_test[i]]}, "
          f"Pred={y_pred_labels[-1]}, Confidence={y_prob[i]:.2f}")

# Count imposters
num_imposters = sum([1 for label in y_pred_labels if label == "Imposter"])
print(f"\nDetected {num_imposters} imposters out of {len(y_test)} test samples")
```

5. Model Optimisation and Evaluation

5.1 Accuracy vs k

Experiments were carried out using PCA with k values of 50, 100, 150, and 200 components. Accuracy was recorded for each setting and plotted against k. Results showed increasing accuracy with larger k values until performance saturated.


```

▶ # Step 8: Calculate accuracy for different k values
# =====
k_values = [10, 20, 50, 100, 150, 200, 250, 300]
accuracies = []

print("Calculating accuracy for different k values...")
for k in k_values:
    print(f"Training with {k} components...")
    pca = PCA(n_components=k, svd_solver='randomized', whiten=True).fit(X_train)
    X_train_pca_k = pca.transform(X_train)
    X_test_pca_k = pca.transform(X_test)

    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train_pca_k, y_train)
    X_train_lda_k = lda.transform(X_train_pca_k)
    X_test_lda_k = lda.transform(X_test_pca_k)

    clf = MLPClassifier(
        random_state=1,
        hidden_layer_sizes=(10, 10),
        max_iter=1000,
        verbose=False # Set verbose to False to avoid printing training progress for each k
    ).fit(X_train_lda_k, y_train)

```

```

✓ ▶ max_iter=1000,
    verbose=False # Set verbose to False to avoid printing training progress for each k
    ).fit(X_train_lda_k, y_train)

y_pred_k = clf.predict(X_test_lda_k)
true_positive_k = np.sum(y_pred_k == y_test)
accuracy_k = true_positive_k * 100 / y_pred_k.shape[0]
accuracies.append(accuracy_k)
print(f"Accuracy for k={k}: {accuracy_k:.2f}%")

print("Done calculating accuracies.")

```

5.2 Final Model Selection

The best results were obtained when k was around 150. This value provided a strong balance between recognition accuracy and computational efficiency.

```

Calculating accuracy for different k values...
Training with 10 components...
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning:
  warnings.warn(
Accuracy for k=10: 41.59%
Training with 20 components...
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning:
  warnings.warn(
Accuracy for k=20: 55.75%
Training with 50 components...
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning:
  warnings.warn(
Accuracy for k=50: 58.41%
Training with 100 components...
Accuracy for k=100: 63.72%
Training with 150 components...
Accuracy for k=150: 69.03%
Training with 200 components...
Accuracy for k=200: 69.03%
Training with 250 components...
Accuracy for k=250: 62.83%
Training with 300 components...
Accuracy for k=300: 17.70%
Done calculating accuracies.

```

6. Results

The trained model demonstrated solid recognition accuracy across test samples. The confusion matrix indicated high performance on most classes, and imposter detection successfully filtered out low-confidence inputs.

7. Advantages & Disadvantages

Advantages:

- Reduces feature dimensionality while preserving face characteristics.
- ANN provides non-linear decision boundaries.
- Imposter detection increases robustness.

Disadvantages:

- Sensitive to changes in lighting and pose.
- A static threshold for imposters may not generalize across datasets.

8. Conclusion

This project successfully demonstrated the use of PCA and ANN for face recognition. The approach achieved good accuracy across varying PCA dimensions, and the inclusion of imposter detection strengthened system reliability.

9. Future Scope

- Extend to convolutional neural networks for better feature extraction.
- Incorporate pose and illumination invariance techniques.
- Deploy the system with a live webcam feed for real-time recognition.

10. Appendix

10.1 Source Code

https://colab.research.google.com/drive/1tx431kudRBPV1pfeytQBeUnTlt4LKCUG?usp=drive_link

10.2 Dataset

Dataset used:

https://github.com/robaita/introduction_to_machine_learning/blob/main/dataset.zip