Name: Shashank Yadav

Id=12342010

Report: Dimensionality Reduction and Classification using LDA and k-NN on MNIST , PCA on Images

Collab
https://colab.research.google.com/drive/1ymc2F8Gxxn2VyenKHdgLf19ObWXEpIp7?usp=sharing

1. Introduction

This report explores the application of Linear Discriminant Analysis (LDA) for dimensionality reduction and its integration with k-Nearest Neighbour's (k-NN) classifier to classify handwritten digits from the MNIST dataset. Observing the change the accuracy after and before applying LDA.

2. Methodology

2.1. Dataset Preparation

-Datase: The dataset used is the MNIST dataset, which consists of 28x28 grayscale images of handwritten digits (0-9). The dataset is fetched fromOpenM.

- Taking the subset: For simplicity and manageable computation, we restrict the dataset to onlydigits 0-4. This is done using a mask that filters out digits 5-9.

- Normalization: Each image is flattened from its 28x28 matrix into a 784-dimensional vector, and the pixel values are normalized to the range [0, 1] using StandardScaler to ensure each feature has a mean of 0 and a variance of

2.2. LDA for Dimensionality Reduction

LDA is used as a dimensionality reduction technique to reduce the data to a lower number of dimensions while preserving the discriminatory power between different classes. The LDA projection transforms the high-dimensional feature space into a lower-dimensional subspace by maximizing the between-class variance while minimizing the within-class variance.

-One-vs-Rest LDA: For each digit (0-4), a One-vs-Rest approach is applied. This involves creating binary labels (1 for the digit of interest, and 0 for all others) and applying LDA for each class to project the data into a single component. The histograms are plotted.

2.3. k-NN Classification

The k-NN classifier is used as a baseline model for classification. It is first applied to the high-dimensional data, then to the LDA-transformed data to evaluate the effect of dimensionality reduction.

Before LDA: The k-NN classifier is applied on the original high-dimensional data, and the accuracy is recorded.

-After LDA: After dimensionality reduction using LDA, the k-NN classifier is applied again on the transformed training and testing sets to evaluate the effect of LDA on classification performance.

3. Observations

3.1. One-vs-Rest LDA Histograms

For each digit (0-4), we plot histograms of the LDA-transformed data. These histograms show the projection of the samples of the current digit versus the other digits. The following observations can be made from these plots:

LDA essentially separates each class from the others by projecting the data onto a 1-dimensional subspace, making it easier to distinguish between the classes.

3.3. k-NN Classification Accuracy

- Before LDA: The accuracy of the k-NN classifier on the original high-dimensional dataset is computed. This acts as a baseline for comparison.

- After LDA: The k-NN classifier is then applied to the data after it has been transformed by LDA. The accuracy is computed again on the transformed dataset.

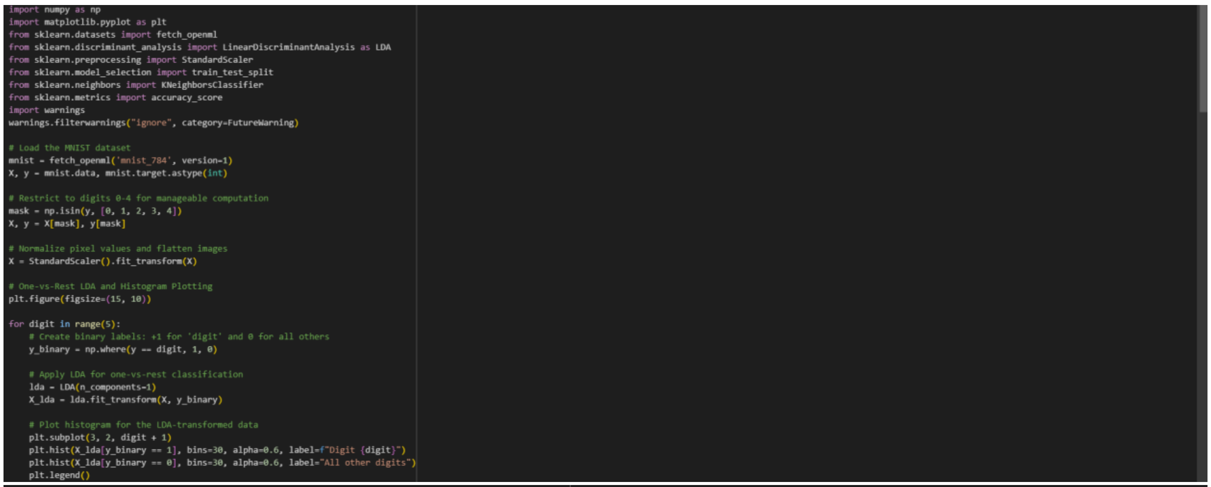
In our case, applying LDA significantly improves the accuracy of the k-NN classifier, as the transformation reduces the dimensionality and removes noise, making the classification task easier. This is a key benefit of dimensionality reduction techniques like LDA.

4. Conclusion

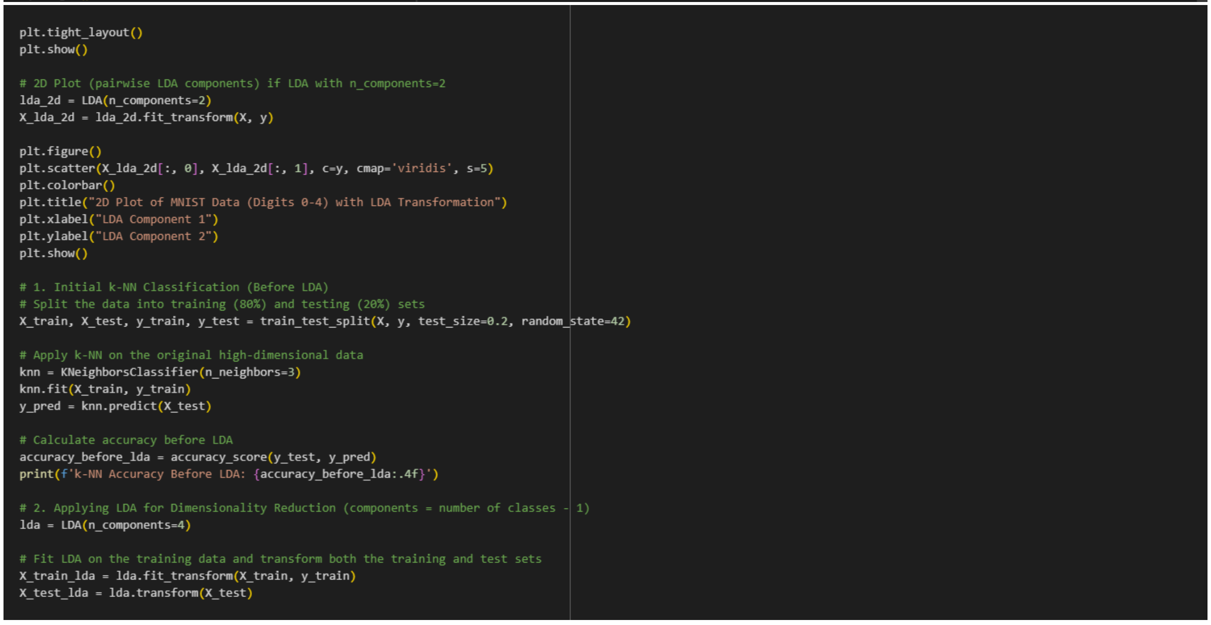
- Dimensionality Reduction: LDA effectively reduces the dimensionality of the MNIST dataset while retaining the most important discriminatory features. The one-vs-rest LDA histograms visually confirm that LDA can separate each digit from the rest of the dataset, even in a 1D projection.

- k-NN Performance: The k-NN classifier performs better after applying LDA for dimensionality reduction, as seen by the increased classification accuracy. This demonstrates that LDA's ability to maximize class separability can improve the performance of classifiers like k-NN, especially in high-dimensional spaces like the MNIST dataset.

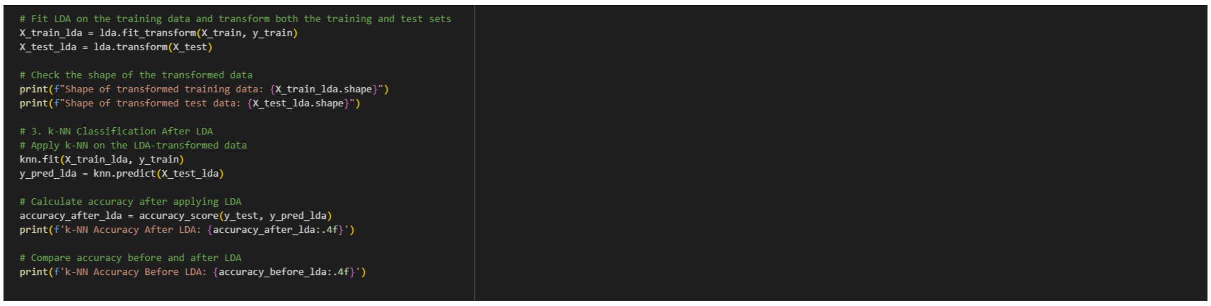
Code:



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target.astype(int)

# Restrict to digits 0-4 for manageable computation
mask = np.isin(y, [0, 1, 2, 3, 4])
X, y = X[mask], y[mask]

# Normalize pixel values and flatten images
X = StandardScaler().fit_transform(X)

# One-vs-Rest LDA and Histogram Plotting
plt.figure(figsize=(15, 10))

for digit in range(5):
    # Create binary labels: +1 for 'digit' and 0 for all others
    y_binary = np.where(y == digit, 1, 0)

    # Apply LDA for one-vs-rest classification
    lda = LDA(n_components=1)
    X_lda = lda.fit_transform(X, y_binary)

    # Plot histogram for the LDA-transformed data
    plt.subplot(3, 2, digit + 1)
    plt.hist(X_lda[y_binary == 1], bins=30, alpha=0.6, label=f"Digit {digit}")
    plt.hist(X_lda[y_binary == 0], bins=30, alpha=0.6, label="All other digits")
    plt.legend()
```

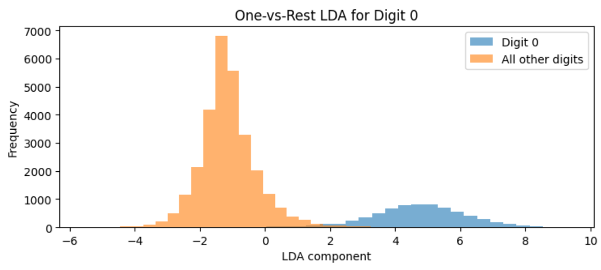


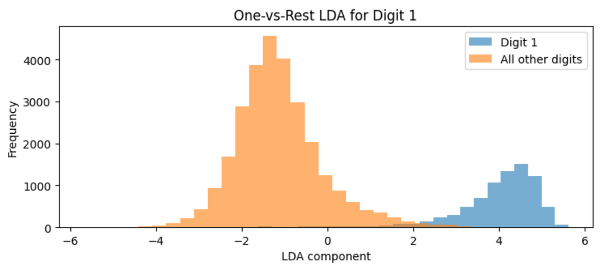
```python
plt.tight_layout()
plt.show()

# 2D Plot (pairwise LDA components) if LDA with n_components=2
lda_2d = LDA(n_components=2)
X_lda_2d = lda_2d.fit_transform(X, y)

plt.figure()
plt.scatter(X_lda_2d[:, 0], X_lda_2d[:, 1], c=y, cmap='viridis', s=5)
plt.colorbar()
plt.title("2D Plot of MNIST Data (Digits 0-4) with LDA Transformation")
plt.xlabel("LDA Component 1")
plt.ylabel("LDA Component 2")
plt.show()

# 1. Initial k-NN Classification (Before LDA)
# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply k-NN on the original high-dimensional data
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Calculate accuracy before LDA
accuracy_before_lda = accuracy_score(y_test, y_pred)
print(f'k-NN Accuracy Before LDA: {accuracy_before_lda:.4f}')

# 2. Applying LDA for Dimensionality Reduction (components = number of classes - 1)
lda = LDA(n_components=4)

# Fit LDA on the training data and transform both the training and test sets
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
```

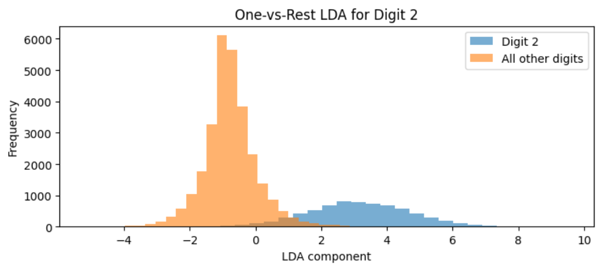


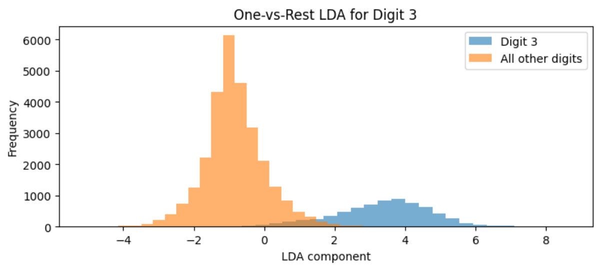
```python
# Fit LDA on the training data and transform both the training and test sets
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# Check the shape of the transformed data
print(f"Shape of transformed training data: {X_train_lda.shape}")
print(f"Shape of transformed test data: {X_test_lda.shape}")

# 3. k-NN Classification After LDA
# Apply k-NN on the LDA-transformed data
knn.fit(X_train_lda, y_train)
y_pred_lda = knn.predict(X_test_lda)

# Calculate accuracy after applying LDA
accuracy_after_lda = accuracy_score(y_test, y_pred_lda)
print(f'k-NN Accuracy After LDA: {accuracy_after_lda:.4f}')

# Compare accuracy before and after LDA
print(f'k-NN Accuracy Before LDA: {accuracy_before_lda:.4f}')
```

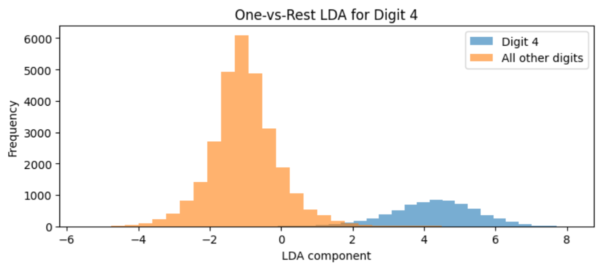
Output:

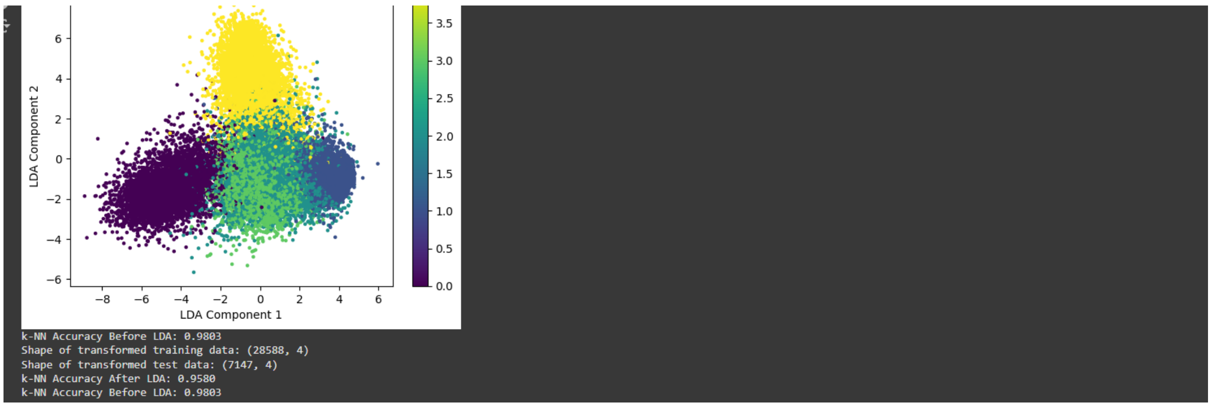












```
k-NN Accuracy Before LDA: 0.9803
Shape of transformed training data: (28588, 4)
Shape of transformed test data: (7147, 4)
k-NN Accuracy After LDA: 0.9580
k-NN Accuracy Before LDA: 0.9803
```

Quenstion 2 : **Dimensionality Reduction using PCA for Face Images**

Collab link

https://colab.research.google.com/drive/1AxdU37Ofy6ifabUIMc6Or1AXirLmrHMt?usp=sharing

1. Introduction

This report demonstrates the application of **Principal Component Analysis (PCA)** for dimensionality reduction and image reconstruction on a dataset of face images. PCA is used to reduce the dimensionality of high-dimensional face images while retaining the most significant features. We perform image reconstruction with varying numbers of principal components to visualize how PCA captures the essence of facial features with fewer dimensions.

2. Methodology

2.1. Image Preprocessing

1. **Loading the Images**:
   - If the image is in colour (RGB format), it is converted to grayscale using rgb2gray.
   - The images are resized to a uniform size of **180x200 pixels**.
2. **Flattening the Images**:
   - Each image is flattened from a 2D array (180x200) into a 1D array (36000 elements) to make it suitable for PCA. Flattening essentially converts each image into a high-dimensional feature vector.
3. **Normalization**:
   - Images are not explicitly normalized in this code, but this step could be added to ensure each pixel value has a mean of zero and variance of one, which could help PCA performance.

2.2. PCA Application for Dimensionality Reduction

PCA is a technique that transforms the data into a new coordinate system such that the first few dimensions (principal components) capture the most variance in the data. In this case, PCA is applied to the flattened face images to reduce their dimensionality while retaining the most important features. The number of principal components is varied to observe the impact on image reconstruction.

2.3. Visualizing Reconstruction with Varying Number of Components

To visualize the impact of dimensionality reduction, the original image is reconstructed using a varying number of PCA components, ranging from 5 to

10 in this experiment. The code iterates through the first image and performs PCA with different component numbers, showing the original image alongside the reconstructed image.

Each image is displayed in two subplots:

1. **Original Image**: The original grayscale image is shown in the first subplot.
2. **Reconstructed Image**: The second subplot shows the image reconstructed using a specific number of principal components. As the number of components decreases, the reconstructed image will show more loss of detail, demonstrating how PCA captures the essential features of the face.

2.4. Code Flow

1. **Loading the images** from subfolders.
2. **Flattening** each image into a 1D array.
3. **Applying PCA** for dimensionality reduction to extract the most important features.
4. **Reconstructing images** using inverse PCA for varying numbers of components.
5. **Plotting** the original and reconstructed images side by side for comparison.

3. Observations

3.1. Image Reconstruction with Varying PCA Components

- **Low Components (5-6)**: When using a small number of components (5 or 6), the reconstructed image shows noticeable blur and loss of detail. This is because only a small portion of the variance in the data is retained, resulting in a simplified version of the original face.
- **Medium Components (7-9)**: As the number of components increases (7, 8, 9), the reconstructed image improves in quality. More facial features are captured, and the images start resembling the original more closely.
- **High Components (10 and above)**: At around 10 components, the reconstructed image is quite close to the original image, but still lacks some fine details.

3.2. Impact of Dimensionality Reduction on Images

The key insight here is that **PCA can effectively reduce the dimensionality** of face images while still preserving most of the essential information needed for recognizing or understanding the faces. Even with only a few principal components.

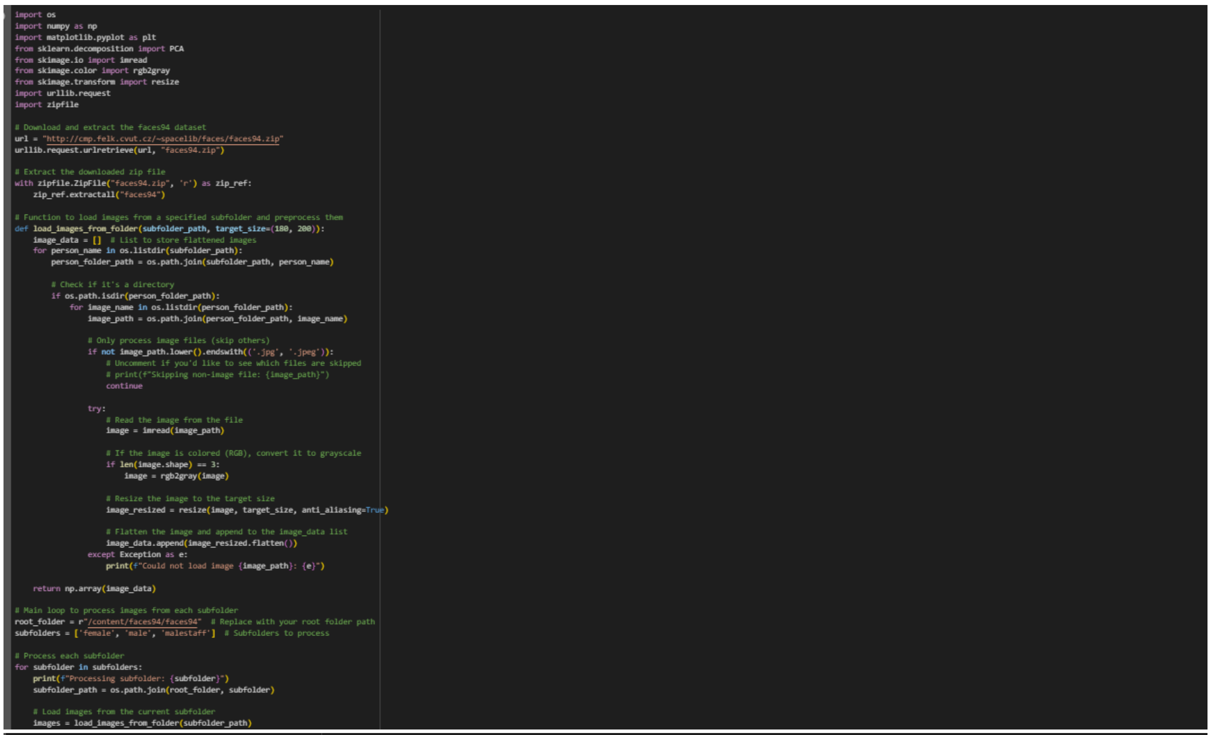
3.3. Quality of Reconstruction

- As the number of components increases, the quality of the reconstruction improves. However, it is clear that even with only a handful of components, PCA can recover the general structure of the face, though finer details are lost.

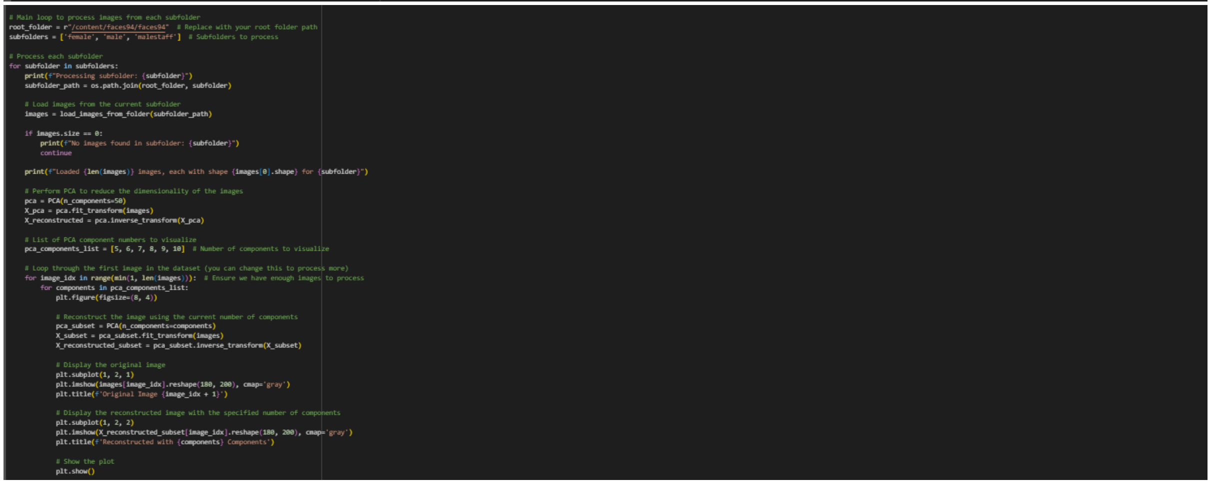
4. Conclusion

- **Dimensionality Reduction with PCA**: PCA proves to be an effective technique for reducing the dimensionality of face images while retaining the most important features.The quality of image reconstruction improves with more principal components.
- **Applications**: This technique can be applied in various computer vision tasks, such as face recognition, where high-dimensional face images need to be compressed and represented more efficiently.

Code:

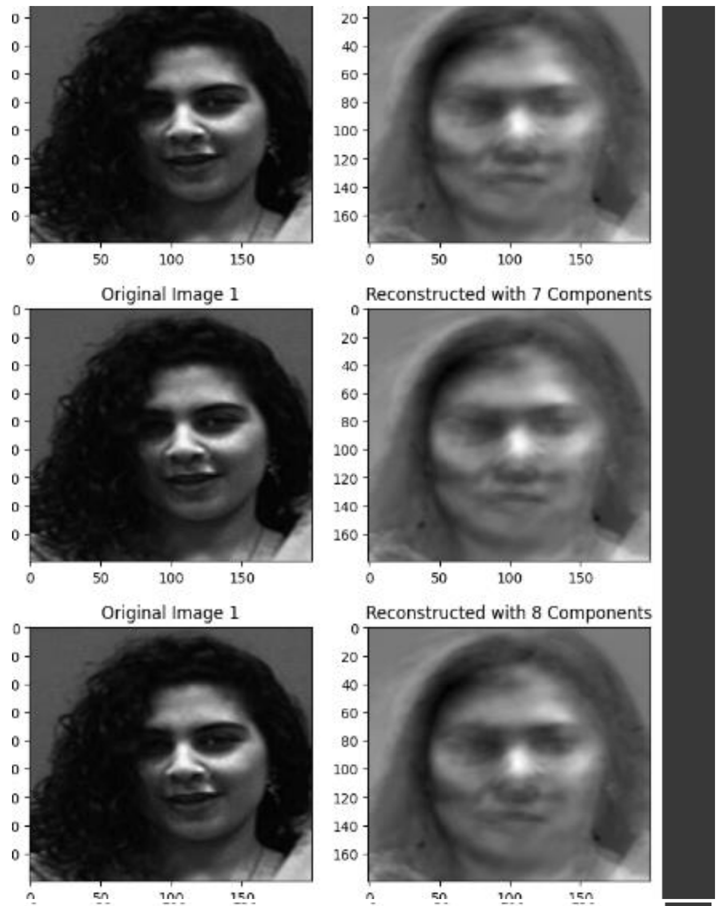


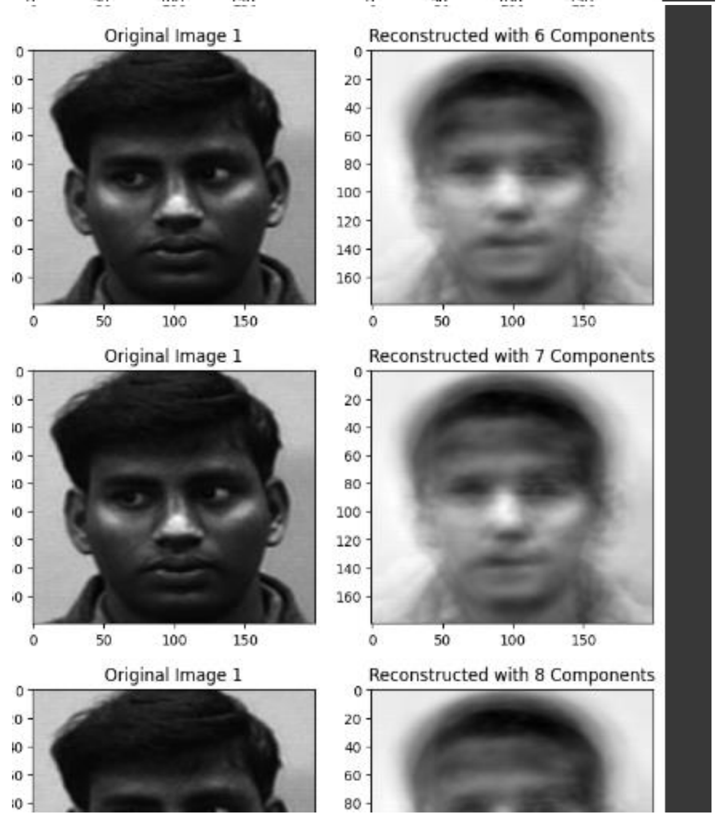
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.transform import resize
import urllib.request
import zipfile

# Download and extract the faces94 dataset
url = "http://cmp.felk.cvut.cz/~spacelib/faces/faces94.zip"
urllib.request.urlretrieve(url, "faces94.zip")

# Extract the downloaded zip file
with zipfile.ZipFile("faces94.zip", 'r') as zip_ref:
    zip_ref.extractall("faces94")

# Function to load images from a specified subfolder and preprocess them
def load_images_from_folder(subfolder_path, target_size=(180, 200)):
    image_data = []  # List to store flattened images
    for person_name in os.listdir(subfolder_path):
        person_folder_path = os.path.join(subfolder_path, person_name)

        # Check if it's a directory
        if os.path.isdir(person_folder_path):
            for image_name in os.listdir(person_folder_path):
                image_path = os.path.join(person_folder_path, image_name)

                # Only process image files (skip others)
                if not image_path.lower().endswith(('.jpg', '.jpeg')):
                    # Uncomment if you'd like to see which files are skipped
                    # print(f"Skipping non-image file: {image_path}")
                    continue

                try:
                    # Read the image from the file
                    image = imread(image_path)

                    # If the image is colored (RGB), convert it to grayscale
                    if len(image.shape) == 3:
                        image = rgb2gray(image)

                    # Resize the image to the target size
                    image_resized = resize(image, target_size, anti_aliasing=True)

                    # Flatten the image and append to the image_data list
                    image_data.append(image_resized.flatten())
                except Exception as e:
                    print(f"Could not load image {image_path}: {e}")

    return np.array(image_data)

# Main loop to process images from each subfolder
root_folder = r"/content/faces94/faces94"  # Replace with your root folder path
subfolders = ['female', 'male', 'malestaff']  # Subfolders to process

# Process each subfolder
for subfolder in subfolders:
    print(f"Processing subfolder: {subfolder}")
    subfolder_path = os.path.join(root_folder, subfolder)

    # Load images from the current subfolder
    images = load_images_from_folder(subfolder_path)
```



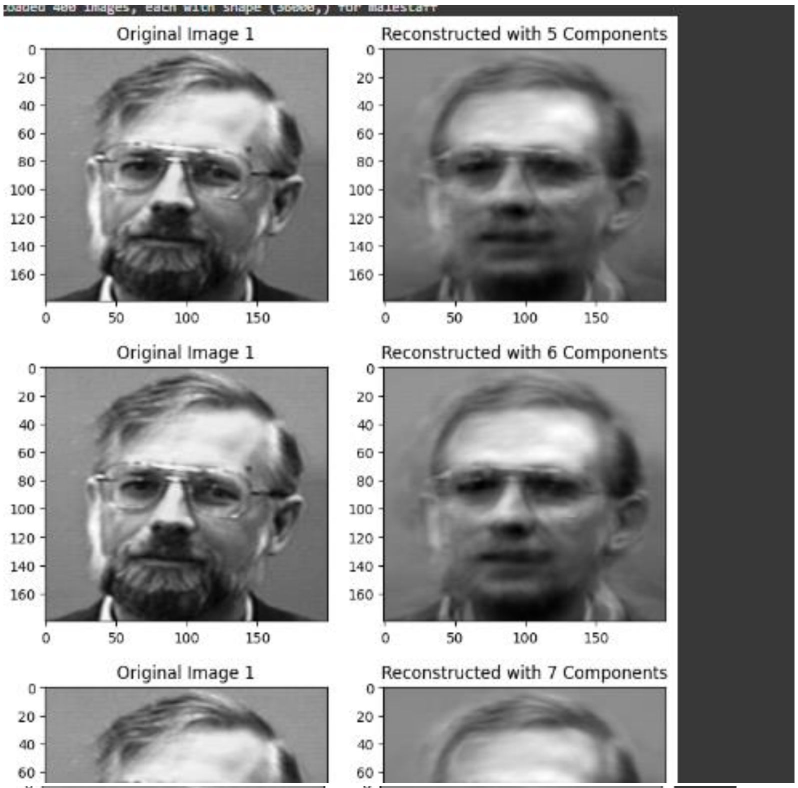
```python
# Main loop to process images from each subfolder
root_folder = r"/content/faces94/faces94"  # Replace with your root folder path
subfolders = ['female', 'male', 'malestaff']  # Subfolders to process

# Process each subfolder
for subfolder in subfolders:
    print(f"Processing subfolder: {subfolder}")
    subfolder_path = os.path.join(root_folder, subfolder)

    # Load images from the current subfolder
    images = load_images_from_folder(subfolder_path)

    if images.size == 0:
        print(f"No images found in subfolder: {subfolder}")
        continue

    print(f"Loaded {len(images)} images, each with shape {images[0].shape} for {subfolder}")

    # Perform PCA to reduce the dimensionality of the images
    pca = PCA(n_components=50)
    X_pca = pca.fit_transform(images)
    X_reconstructed = pca.inverse_transform(X_pca)

    # List of PCA component numbers to visualize
    pca_components_list = [5, 6, 7, 8, 9, 10]  # Number of components to visualize

    # Loop through the first image in the dataset (you can change this to process more)
    for image_idx in range(min(1, len(images))):  # Ensure we have enough images to process
        for components in pca_components_list:
            plt.figure(figsize=(8, 4))

            # Reconstruct the image using the current number of components
            pca_subset = PCA(n_components=components)
            X_subset = pca_subset.fit_transform(images)
            X_reconstructed_subset = pca_subset.inverse_transform(X_subset)

            # Display the original image
            plt.subplot(1, 2, 1)
            plt.imshow(images[image_idx].reshape(180, 200), cmap='gray')
            plt.title(f'Original Image {image_idx + 1}')

            # Display the reconstructed image with the specified number of components
            plt.subplot(1, 2, 2)
            plt.imshow(X_reconstructed_subset[image_idx].reshape(180, 200), cmap='gray')
            plt.title(f'Reconstructed with {components} Components')

            # Show the plot
            plt.show()
```

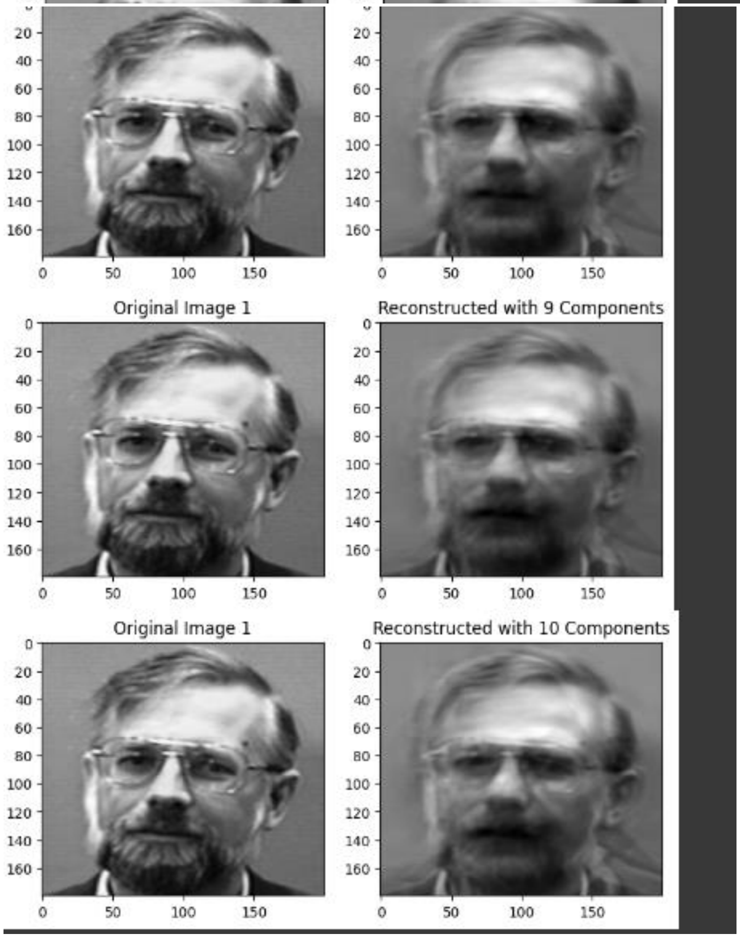
Output:


Original Image 1
Reconstructed with 7 Components
Original Image 1
Reconstructed with 8 Components


Original Image 1
Reconstructed with 6 Components
Original Image 1
Reconstructed with 7 Components
Original Image 1
Reconstructed with 8 Components

```
oaded 400 images, each with shape (36000,) for malestaff
```


Original Image 1
Reconstructed with 5 Components
Original Image 1
Reconstructed with 6 Components
Original Image 1
Reconstructed with 7 Components


Original Image 1
Reconstructed with 9 Components
Original Image 1
Reconstructed with 10 Components