Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.Doi Number

Facial Data Analysis through Principal Component Analysis and Image Reconstruction

PRABIN BOHARA[1], PRABIN SHARMA POUDEL[2], AND SANTOSH PANDEY[3]
[1]Institute of Engineering, Thapathali Campus, Thapathali, Nepal
[2]Institute of Engineering, Thapathali Campus, Thapathali, Nepal
[3]Institute of Engineering, Thapathali Campus, Thapathali, Nepal

*All authors contributed equally

ABSTRACT Grappling with the complexities of high-dimensional datasets, this report delves into the usefulness of Principal Component Analysis (PCA) in tackling highly redundant dataset information such as images by enabling efficient feature extraction, data compression, and visualization, enhancing model training and performance. The work presents the application of PCA to a facial dataset, comprising images of forty-four individuals. Through systematic pre-processing, dimensionality reduction, and eigen-decomposition, we demonstrate PCA's effectiveness in retaining essential facial features while reducing data complexity. The reconstructed images using varying Principal Components were validated through human analysis and quantified by reconstruction loss assessment. This approach holds promise in training machine learning models efficiently while maintaining data integrity. This study underscores PCA's potential for enhancing facial image analysis, fostering better model interpretability, and driving informed decision-making.

INDEX TERMS Dimensionality Reduction, Principal Component Analysis, Reconstruction Loss

I. INTRODUCTION

In an era of data proliferation, resulted by the changing digital landscape, has presented us with vast and intricate datasets that hold the potential for valuable insights. In simple terms, there hasn't been more data like this before, and it's only likely to grow even more. While this presents us with countless opportunities, it brings an overwhelming amount of confusion along. Amid this data deluge, the Principal Component Analysis (PCA) technique emerges as a beacon of data understanding. By addressing the challenges posed by high-dimensional datasets, PCA empowers us to reveal the underlying trends and patterns that shape our data universe. PCA achieves this by transforming the original dataset into a reduced-dimensional space, preserving essential information while simplifying its representation. This reduction not only streamlines analysis but also encapsulates the fundamental structure of the data. As such, PCA serves as a formidable ally in navigating the complexities of modern data analysis, helping us uncover meaningful insights amidst the digital abundance. Building on a robust methodology that encompasses data pre-processing, covariance matrix computation, eigenvalue-eigenvector decomposition, and dimensionality reduction, we embark on a journey to harness the power of PCA.

In this report, our focus is on the application of PCA to an intriguing realm: facial datasets. The dataset consists of front view of the faces of forty-four students from Computer Engineering department of Thapathali Campus, Kathmandu. The dataset consists of six female and thirty-eight male population, which might affect the latent dimensions that determine the facial features. By leveraging PCA we seek to determine the most relevant features present in the dataset we have collected. Furthermore, our exploration delves into the potential of PCA in understanding the intricate characteristics of facial images. Since facial datasets contain highly redundant features, PCA is particularly useful in this application, for dimensionality reduction and further facial recognition and analysis that might follow. As we journey through the intricacies of dimensionality reduction and data representation, we aim to shed light on the potential of PCA in advancing facial analysis, recognition, and our broader understanding of data-driven insights.

II. INTRODUCTION

A. THEORY

PCA operates by capturing the most significant dimensions, known as principal components, that encapsulate the majority of variance present in the dataset. These components are determined based on the eigenvalues and eigenvectors of the covariance matrix computed from the dataset. By ranking the eigenvalues in descending order, the principal components are selected, resulting in a compressed representation of the original data. The concept of variance plays a crucial role in PCA. Variance measures the spread of data points around their mean and reflects the dataset's overall variability. In PCA, we aim to maximize the variance captured by the principal components. This is achieved by considering the eigenvalues associated with these components. Higher eigenvalues correspond to dimensions that contain more variance and are therefore prioritized for inclusion in the reduced-dimensional space.

The covariance matrix is central to PCA. It quantifies the relationships and interactions between different dimensions in the dataset. By computing the covariance matrix, we obtain insights into how dimensions co-vary. The eigen decomposition of the covariance matrix involves calculating the eigenvalues and eigenvectors. Eigenvectors represent the directions along which the data exhibits maximum variance, while eigenvalues quantify the amount of variance along these directions. Upon obtaining the eigenvectors, the principal components emerge as the new basis vectors for the transformed data space. These components provide directions that best capture the data's variability. Each principal component is orthogonal to others, indicating that the data is uncorrelated along these dimensions. This orthogonality simplifies interpretation and allows for efficient dimensionality reduction.

PCA involves projecting the original data points onto the selected principal components. This projection transforms the data from its high-dimensional space to a lower-dimensional space spanned by the principal components. The transformed data retains essential features while discarding dimensions with lower variance. In the context of facial datasets, PCA finds profound utility. Eigenfaces refer to the principal components extracted from facial images. These components correspond to characteristic facial features and capture the most distinctive aspects of the dataset. By using eigenfaces, facial recognition and analysis can be conducted efficiently, aiding in various applications such as identity verification and emotion detection. The proportion of variance explained by each principal component is determined by the ratio of the corresponding eigenvalue to the sum of all eigenvalues. This proportion signifies the amount of information preserved when the dataset is projected onto a specific principal component. By selecting a subset of principal components that collectively explain a significant portion of variance, we achieve dimensionality reduction while retaining data integrity.

B. SYSTEM BLOCK DIAGRAM

The system architecture above shows various blocks in sequence. Facial dataset of forty-four students were collected and visualized. Most images were of the size 128*128 whereas some images were of different sizes, which were resized for uniformity. Since the images were in RGB format, this would imply that there would be 3 different channels, each holding their respective pixel values. This would mean that there would be more pixels per image, which is unnecessary for our case. To eliminate this, the images were converted to Grayscale format, and the image then only had one channel, simplifying the operation by three times.

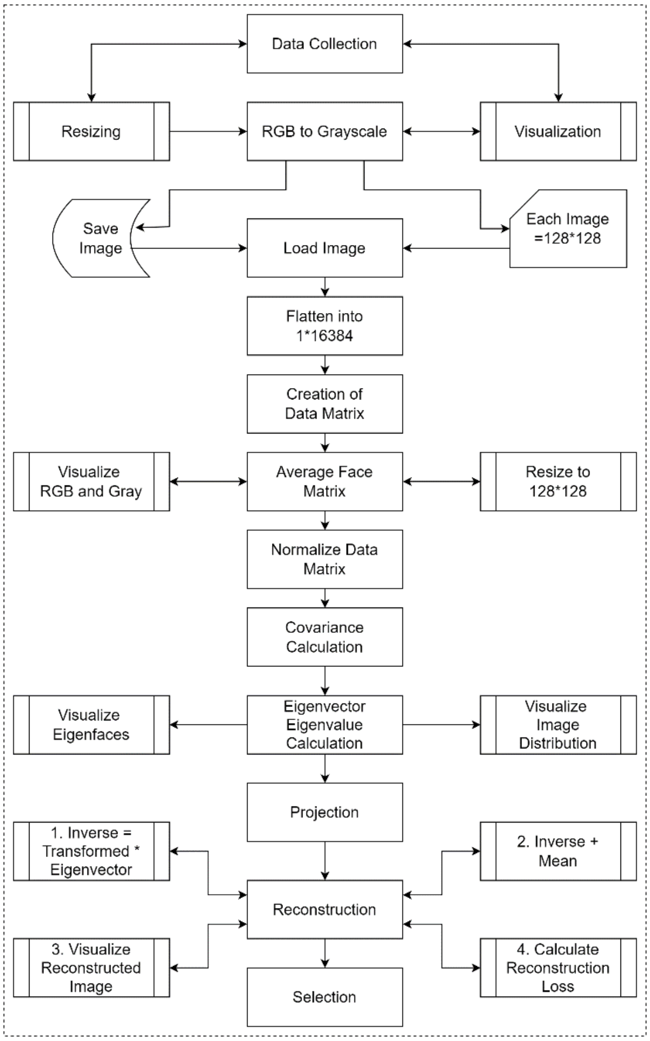


FIGURE 1. System Block Diagram

The pixel values were flattened, and all the flattened images were stacked vertically to create a data matrix of size 44*16384. Now with the column-wise operation, average face matrix was calculated. This matrix was used for various purposes, such as visualization and normalization of the data matrix. Then covariance matrix was calculated, which was followed by the calculation of eigenvectors and eigenvalues. Top 10 eigenvectors were used for the purpose of obtaining eigenfaces, which were visualized. Projection was done using matrix multiplication between the row feature vector and row matrix of the normalized data matrix. For the purpose of reconstruction, the inverse of the projection was done, followed by the addition of the mean into the normalized data matrix. Some samples were visualized to check the optimum number of PCA that was helpful in regenerating the original image. Total reconstruction loss was also calculated to check the loss in image information during the projection and reconstruction process, by comparing the reconstructed image with the original image. The minimum number of Principal Components that could result in the lowest reconstruction loss was chosen as the optimum number of Principal Components.

C. INSTRUMENTATION

In our implementation of Principal Component Analysis (PCA) for facial image analysis, we employed a range of libraries and tools to streamline data manipulation, visualization, and mathematical operations. These tools included:

Pandas: A versatile Python library that facilitated efficient data manipulation and analysis. It offers efficient data structures, like Data Frame that help us in easy indexing and manipulation of data. We can harness Pandas for loading, reading, and pre-processing the dataset. It aided in tasks like inspecting the data and performing data scaling. Pandas streamlined the extraction of features and descriptive statistics, contributing to our comprehensive data understanding. Additionally, Pandas was used to select specific columns, filter rows based on conditions, and perform descriptive statistics on the data.

Matplotlib: A powerful data visualization tool that aided in transforming complex data into insightful visuals. It helps us gain insights, communicate findings, and present results. It offers flexible options for creating line plots, scatter plots, bar plots, histograms, heat maps, and more. We used Matplotlib to create charts to visualize different aspects of the data and analyze patterns and relationships. These visualizations allowed us to uncover patterns, correlations, and trends within the facial image dataset.

NumPy: A fundamental library for numerical computations that played a crucial role in our

PCA implementation. We capitalized on NumPy for mathematical operations on arrays, matrix manipulations, and linear algebra computations. It was essential for tasks such as calculating eigenvalues and eigenvectors, which are pivotal in PCA's dimensionality reduction.
Scikit-image (Skimage): This library was instrumental in converting RGB images to grayscale format, a prerequisite for our facial dataset analysis. It simplified the process of image loading and manipulation.
OS and PIL: We employed the 'OS' library to interact with the operating system, specifically to read image files from directories. The Python Imaging Library (PIL) enabled us to work with images by converting them into arrays, an integral step in PCA pre-processing.
The amalgamation of these libraries facilitated a seamless integration of data visualization, manipulation, and mathematical computations in our PCA analysis. This instrumentation significantly contributed to our ability to comprehend the dataset, visualize outcomes, and glean meaningful insights from the dimensionality reduction process.

D. WORKING PRINCIPLE

The working principle involves a pipeline of steps that are provided below:

- Dataset Collection and Pre-Processing:

The facial images of forty-four students were collected for analysis. The images were of varying sizes; thus, they had to be resized to a uniform size of 128x128 pixels. The images were in RGB format, and since there were three red, green and blue channels having their own set of pixel values, each image would be of the size 128x128x3 = 49152. This complexity in size can be avoided by three folds if they are to be converted into a different format that has a single channel. Thus, we converted the images to Grayscale format and the size of each image was reduced to 128x128 pixels.

FIGURE 2. **Sample Images in RGB format**

FIGURE 3. **Sample Images in Grayscale format**

- Data Matrix Creation:

All the 128x128 images were flattened into a 1x16384 vector. This not only simplified the image by reducing the dimension but also helped in the creation of the data matrix. On top of one flattened image, others were vertically stacked to form a data matrix of size 44x16384. Here, the columns represented all the pixels of an image, whereas the rows represented the forty-four images who had been stacked on top of each other.

- Average Face Matrix and Data Matrix Normalization:

The average face matrix was calculated by averaging all the images in the data matrix. The column wise average was computed and stored for each column. This again reduced the matrix size to 1x16384. Using this average face matrix, the normalization of the above data matrix is done to center the data. Again, the matrix was reshaped to 128x128 where each row and column contained the pixel values for an average person. It was visualized, providing a representative image of the dataset.

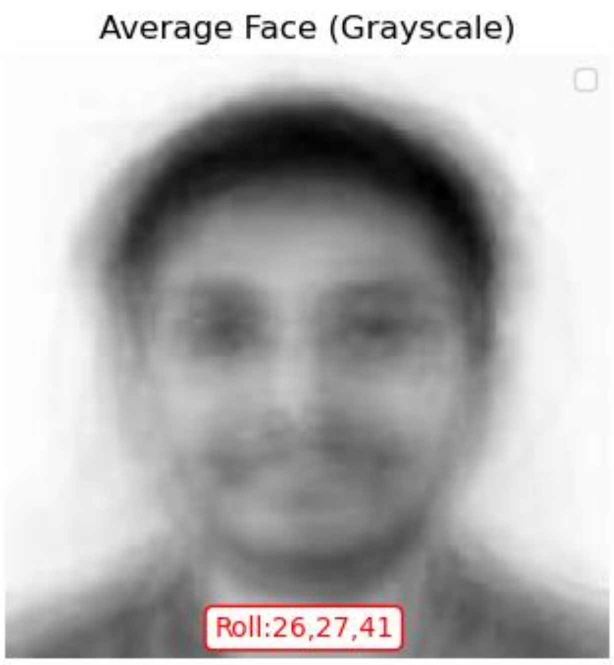


FIGURE 4. Average Face in Grayscale

- Covariance, Eigen Vector, and Eigenvalues Calculation:

The covariance matrix was computed based on the pre-processed dataset. The covariance matrix captures the relationships between the variables and provides information about the variance and covariance of the data. This step involved calculating pairwise covariances between the features in the dataset. The covariance matrix can be used to identify highly correlated features. The covariance matrix is then decomposed into its eigenvalues and eigenvectors. This step is performed using numerical methods such as the eigenvalue decomposition provided by libraries like NumPy. The eigenvalues represent the variance explained by each eigenvector, while the eigenvectors define the principal components.

- Visualization of Eigen Faces:

The top 10 eigenvectors were selected to plot the eigenfaces. The eigenvectors represent that they contribute the most to the dataset's variance. In image's terms, the highly redundant parts of the pixels do not amount to much of variance. Unique facial features such as beard, long hair, spectacles could capture the interestingness in the analysis. The visualization of these eigenfaces depicted the dominant facial features captured by these principal components.

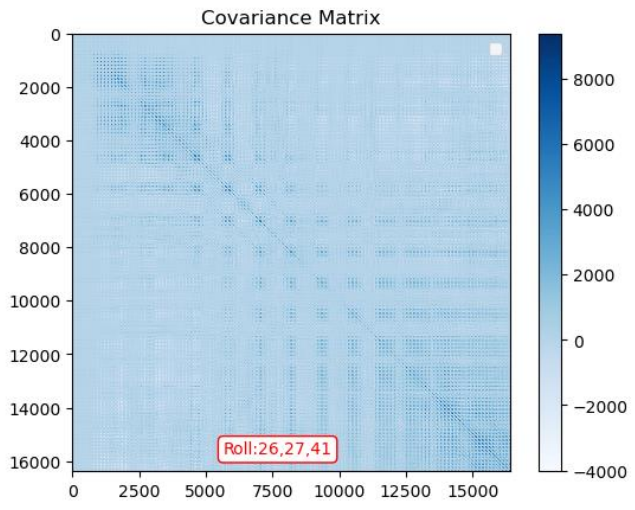


FIGURE 5. Correlation of Features

- Projection and Reconstruction:

The principal components are selected based on the eigenvalues. The eigenvectors with higher eigenvalues capture more variance in the data and are chosen as the principal components. The original data matrix was projected onto the top eigenvectors to transform the dataset into a reduced-dimensional space. For reconstruction, the inverse transformation was performed by multiplying the projected data with the selected eigenvectors' transpose. The average matrix was added to the reconstructed data to account for the normalization process, since this is the reverse of all the steps that were leading up to this step.

- Optimal Principal Components and Reconstruction Loss:

The reconstructed images were visualized using varying numbers of principal components. The reconstruction loss was evaluated by comparing the original image with its reconstructed counterpart. The minimum number of principal components were determined by the visual inspection as well as using the math that yielded the lowest reconstruction loss, representing the optimal dimensionality reduction.

This pipeline illustrated the step-by-step process of applying Principal Component Analysis (PCA) to facial image data. Each stage involved mathematical computations, visualization, and transformations that collectively lead to a reduced-dimensional representation of the dataset while preserving its essential features. The reconstruction loss assessment ensures that the optimal number of principal components is chosen for efficient data compression and model performance enhancement.

III. RESULTS

The forty-four images, each of size 128x128, after flattening and stacking gave us a data matrix which was used to compute the average face matrix and visualize the average face. The figure is particularly helpful in drawing a rough conclusion of what the majority of the faces look like. Similarly, the covariance matrix, with a shape of 16384x16384, showcased the correlations between pixel values across all images. As displayed in the covariance matrix plot, pixels capturing irrelevant background details exhibited low correlations, indicating their potential for elimination. This is useful in retaining only the most relevant features that contribute significantly to the dataset's variance.

The plot of the top 10 eigenfaces illustrated the principal components capturing distinct facial features from the dataset. Each eigenface is computed by using the corresponding eigenvector associated with the top eigenvalues. Eigenfaces signify specific facial variations such as lighting conditions, poses, and expressions, while the dominant eigenfaces highlight the most significant variations among the images.

The 3D graph portrayed the projection of 44 facial images onto the top three principal components. Each axis represented one principal component, and images' positions are highlighted as their representation in this reduced-dimensional space. This visualization showcased the clustering and distribution of facial images in relation to these top principal components

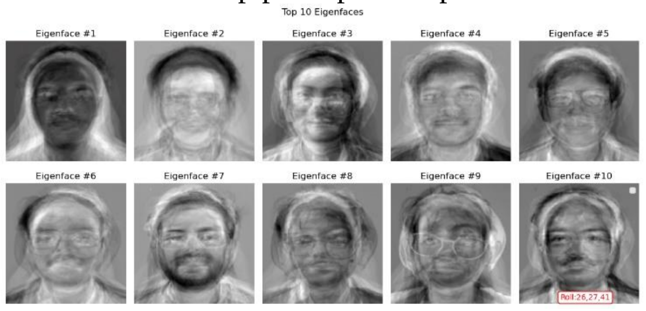


FIGURE 6. Top 10 Eigen Faces

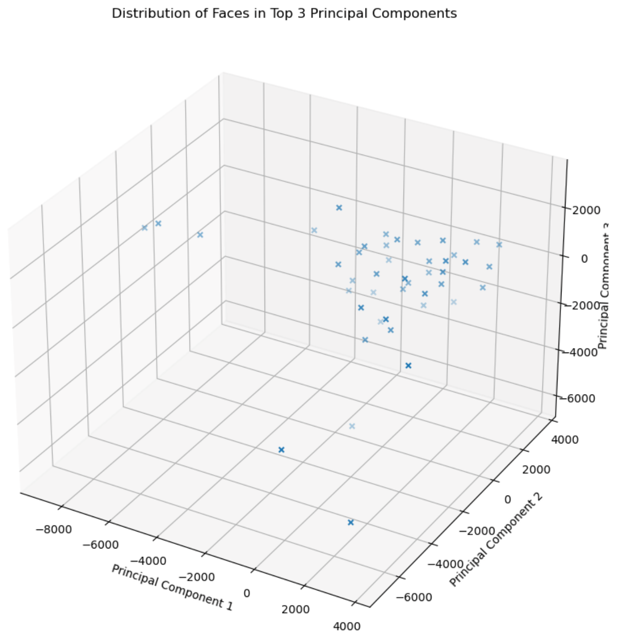


FIGURE 7. Image Distribution in Top 3 Components

The Scree plot illustrated the proportion of variance explained by each eigenvalue in descending order. In the plot, the cumulative variance initially increased steeply, representing the high information content captured by the initial principal components. The curve then levelled off, indicating that later components contributed less to the overall variance and could potentially be discarded for dimensionality reduction. Likewise, the explained variance for each of the individual Eigen values was shown by the blue line. It showed that the variance explained by the descending order of Eigen values kept on decreasing until it saturated after about the value of 40.

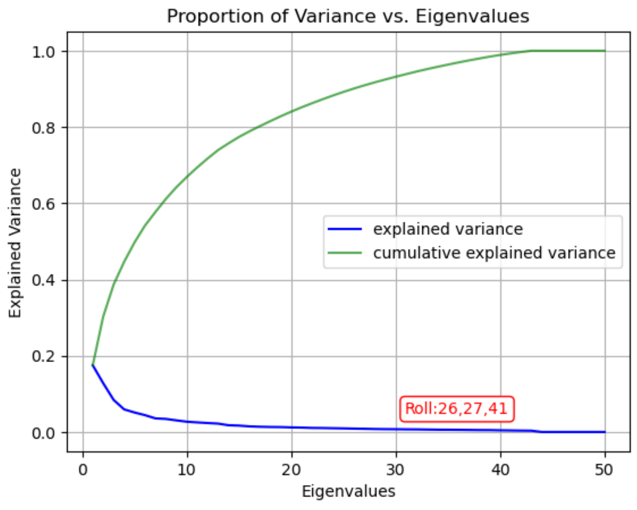


FIGURE 8. Scree Plot for Explained Variance

Reconstructed images using different numbers of principal components (11, 21, 31, 41, 61) were observed and analyzed. Among these, the images reconstructed with 41 principal components seemed to best retain the key facial features while minimizing noise and distortion. This observation implied that using 41 principal components optimally balanced information retention and dimensionality reduction for these facial images.

The reconstruction loss curve (MSE vs Principal Components) provided insights into the trade-off between data compression and information loss. Initially, as the number of principal components increased, the reconstruction loss rapidly decreased, indicating significant information preservation. The curve eventually levelled off, signifying the point where further addition of principal components contributed marginally to reduction in loss, highlighting the optimal number of components. Both the visual inspection of the reconstructed data and the reconstruction loss graph suggested that the optimum number of Principal Components was 41.

IV. DISCUSSION AND ANALYSIS

The results obtained by applying Principal Component Analysis on images and Reconstruction of the original image showed promising results.

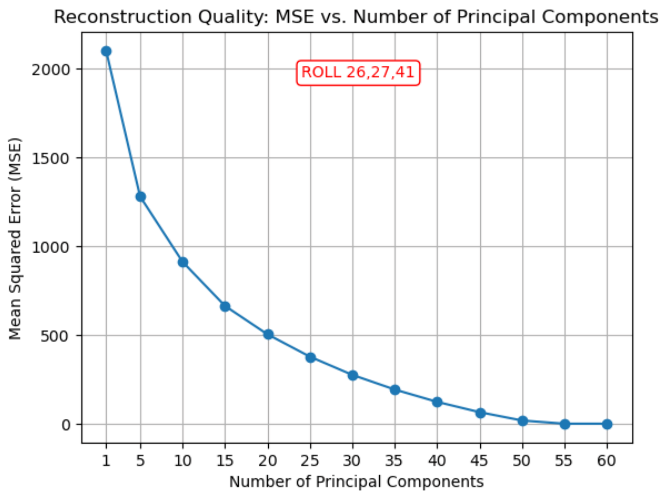


FIGURE 9. Reconstruction Loss (MSE)

However, it can be seen that the Scree plot and the Reconstruction loss curve saturates are two different points in the X-axis. The former indicates the optimum number of Principal Components (PC) should be about 40, whereas the Reconstruction loss curve indicates the optimum number of PC should be about 50. The disparity in saturation points arises due to the different objectives of the scree plot and the MSE curve. The scree plot focuses on variance explanation, which may reach saturation earlier as it considers capturing dataset variability in fewer components. The MSE curve is concerned with image reconstruction accuracy, which might require a few more components to capture finer details. The scree plot guides us towards capturing most of the dataset's variability with fewer components, useful for visualization. The MSE curve emphasizes accurate image reconstruction, which might require slightly more components for finer details. The decision on the number of optimum principal components should be made thoughtfully, considering the trade-off between dimensionality reduction and data representation quality. The decision depends on the application's goal. If visualization or data compression is the priority, the scree plot can guide towards a smaller number of components. If accurate image reconstruction for applications like facial recognition is crucial, the MSE curve's saturation point might be more informative. Practical implementation might involve finding a balance between both approaches and possibly considering user feedback or specific application requirements. In our case, the other validation tool would be for the users to notice at the reconstructed image and decide the optimum number of components. If the user's decision supports that indicated by the scree plot, the optimum components suggested by the scree plot should be chosen and likewise for the given MSE curve.

Some images needed fewer Principal components to be reconstructed within an acceptable quality whereas others needed some more components as seen from the presented results. Some images have unique features and such images with more variability often require fewer components for accurate reconstruction, while images with less variability or uniform features might need more components to be accurately captured. For a dataset with two kind of extreme dataset, one that needs fewer components and the other needs more components, the choice of optimum principal components becomes difficult. The optimum number of principal components is a trade-off between preserving meaningful information and reducing dimensionality. The choice of the optimum number depends on the application's requirements. A balance between accurate reconstruction and dimensionality reduction is essential. Selecting too few components might lead to significant information loss, while too many might introduce noise and redundancy. Suppose ff half the images need 40 components and the other half need 25, an average might not be the most accurate approach. Averaging might compromise the quality of image representation for both types, as it's not tailored to their distinct reconstruction requirements. For tasks like facial recognition, it is wiser to prioritize accurate reconstruction for individual images. In fact, a dynamic approach, like selecting the number of components on a per-image basis, might yield superior results.

The requirement for about 40 principal components for acceptable reconstruction could indeed be attributed to the curse of dimensionality. In high-dimensional spaces, data points tend to become sparse due to the vast number of possible combinations of feature values. This sparsity can make it difficult for a smaller number of principal components to capture the significant variations present in the data. Reducing the resolution can sometimes help to address this problem because it will create a matrix of smaller size, which is not only beneficial from computational standpoint but also beneficial in capturing high content information using fewer principal components during PCA. In this case, reducing image resolution is not an option, so feature engineering can help create more informative features that might be less sparse. Likewise, other dimensionality reduction techniques such as Factor Analysis, Linear Discriminant Analysis (LDA) can be used on top of PCA.

V. CONCLUSION

In conclusion, this project embarked on a comprehensive exploration of Principal Component Analysis (PCA) applied to facial datasets, unravelling its potential in addressing intricate challenges of high-dimensional data analysis. Employing a meticulous pipeline encompassing data pre-processing, covariance matrix computation, eigenvalue-eigenvector decomposition, and dimensionality reduction, we navigated through an array of images with the aim of enhancing data comprehension and unveiling latent patterns. By harnessing the power of PCA, the dimensionality of the facial dataset was effectively reduced, facilitating both visualization and model training while retaining substantial information integrity. The distinctive strength of PCA in compressing data and augmenting visualization rendered insights into the rich interplay of features. This endeavor underscores the versatile capabilities of PCA in navigating high-dimensional realms, accentuating its role in knowledge extraction, model improvement, and data-driven decision-making processes. Ultimately, this project contributes to the continual advancement of data analysis methodologies, fostering valuable implications for diverse applications.

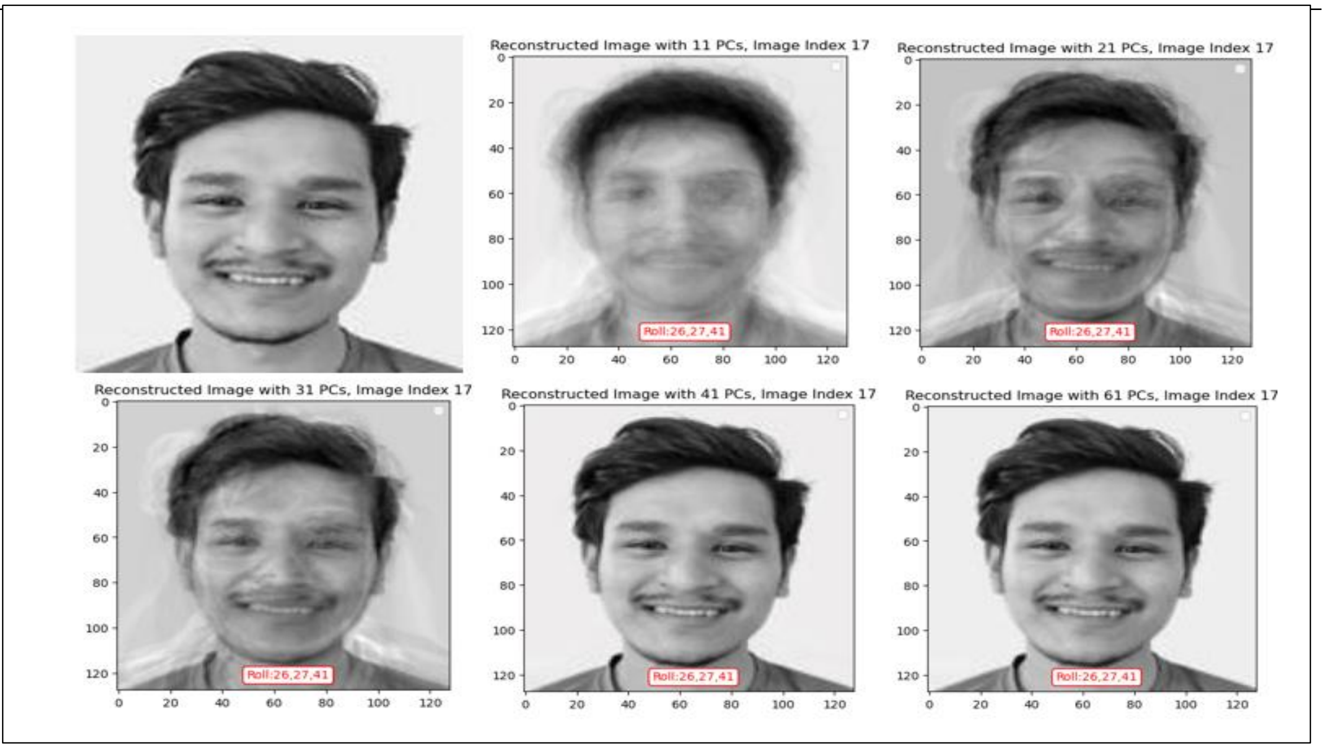

Reconstructed Image with 11 PCs, Image Index 17
Reconstructed Image with 21 PCs, Image Index 17
Reconstructed Image with 31 PCs, Image Index 17
Reconstructed Image with 41 PCs, Image Index 17
Reconstructed Image with 61 PCs, Image Index 17
Roll:26,27,41

FIGURE 10. Original vs Reconstructed Image sample 1

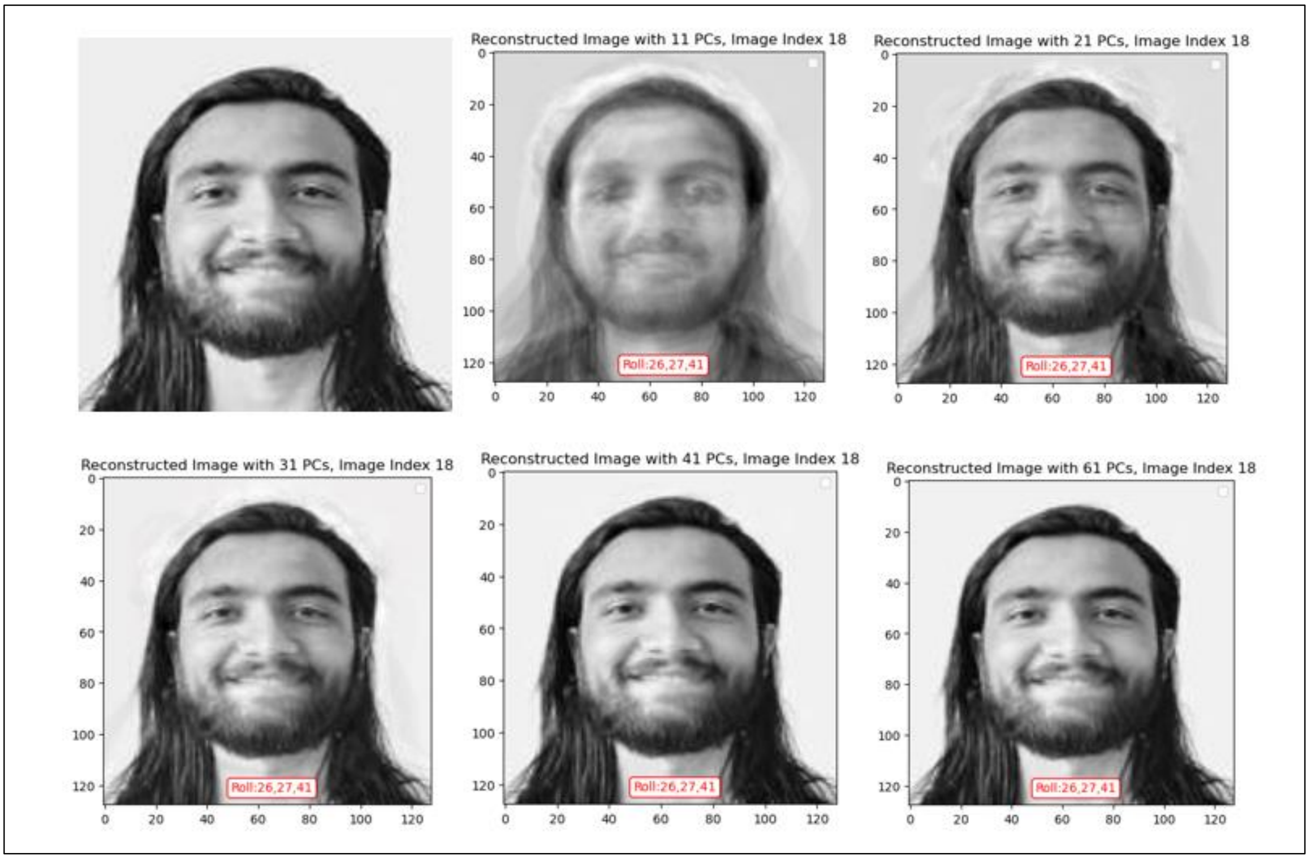

Reconstructed Image with 11 PCs, Image Index 18
Reconstructed Image with 21 PCs, Image Index 18
Reconstructed Image with 31 PCs, Image Index 18
Reconstructed Image with 41 PCs, Image Index 18
Reconstructed Image with 61 PCs, Image Index 18
Roll:26,27,41

FIGURE 11. Original vs Reconstructed Image sample 2

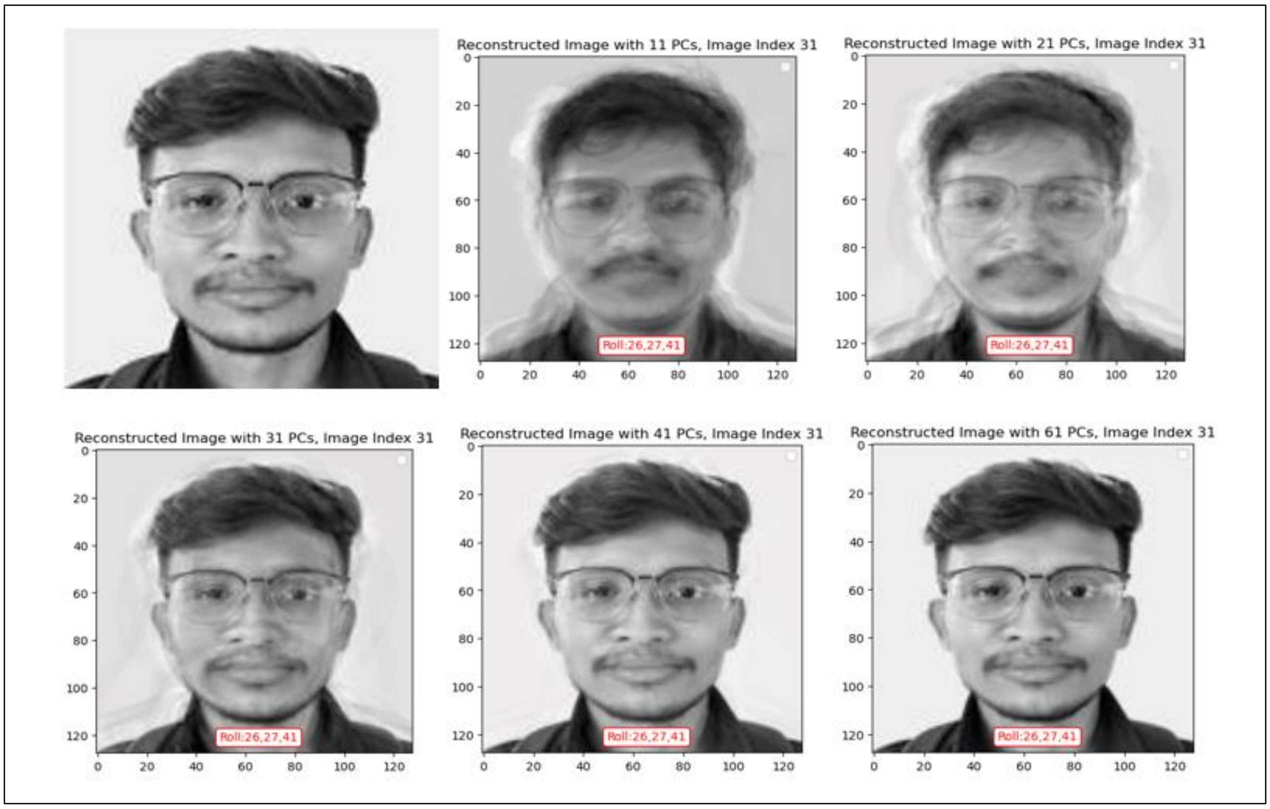
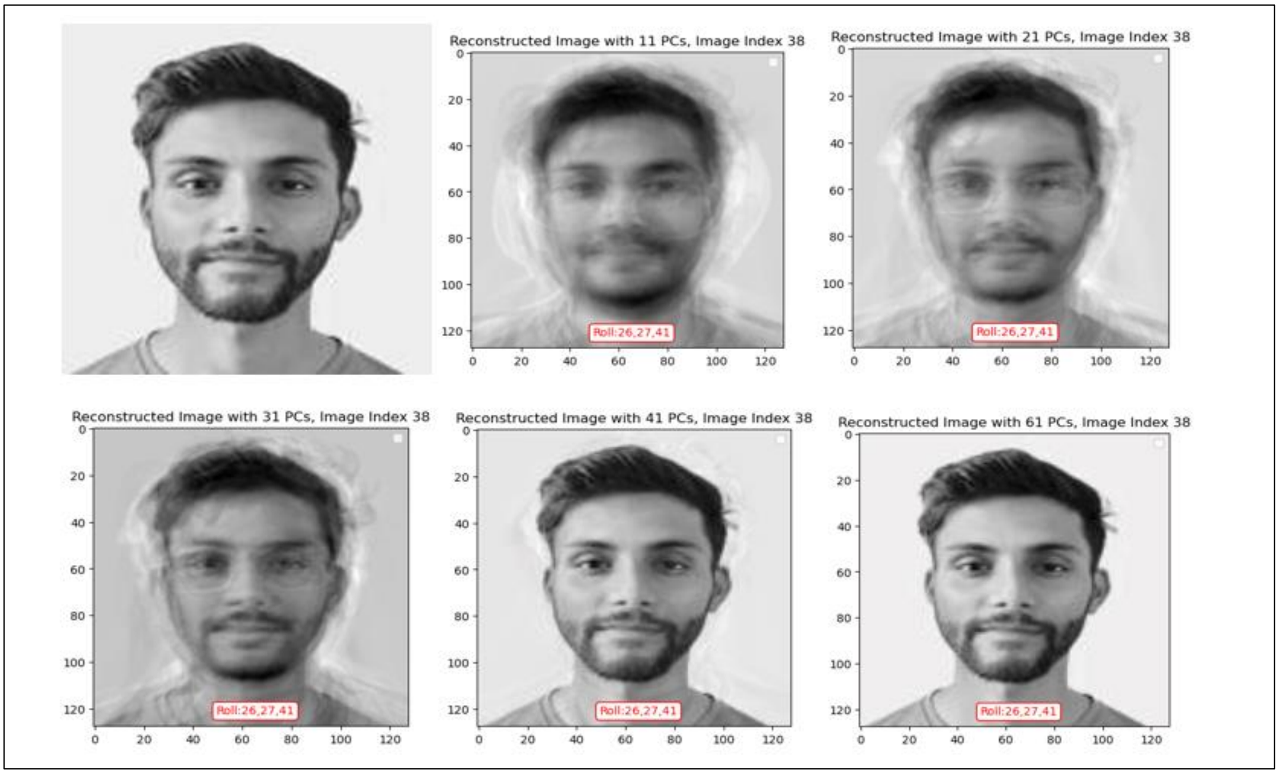

Reconstructed Image with 11 PCs, Image Index 31
Reconstructed Image with 21 PCs, Image Index 31
Reconstructed Image with 31 PCs, Image Index 31
Reconstructed Image with 41 PCs, Image Index 31
Reconstructed Image with 61 PCs, Image Index 31
Roll:26,27,41

FIGURE 12. **Original vs Reconstructed Image sample 3**


Reconstructed Image with 11 PCs, Image Index 38
Reconstructed Image with 21 PCs, Image Index 38
Reconstructed Image with 31 PCs, Image Index 38
Reconstructed Image with 41 PCs, Image Index 38
Reconstructed Image with 61 PCs, Image Index 38
Roll:26,27,41

REFERENCES

[1]] Abdi, H., & Williams, L. J. (2010). Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4), 433-459. doi:10.1002/wics.101.
[2] I. T. Jolliffe, "Principal component analysis and factor analysis," Principal Component Analysis, pp. 115-128, 1986.
[3] K. Pearson, "On lines and planes of closest fit to systems of points in space," Philosophical Magazine, vol. 2, no. 11, pp. 559-572, 1901.
[4] H. Abdi and L. J. Williams, "Principal component analysis," Wiley Interdisciplinary Reviews: Computational Statistics, vol. 2, no. 4, pp. 433-459, 2010.
[5] A. Géron, "Principal Component Analysis (PCA) Explained," Medium, [Online]. Available: https://towardsdatascience.com/principal-componentanalysis-pca-explained-and-implemented-eeab7cb73b72.

PRABIN BOHARA is an enthusiastic individual currently pursuing a Bachelor's degree in Computer Engineering, set to graduate in 2024. He has a genuine passion for coding, web technologies, and machine learning. Prabin's primary focus revolves around AI research and exploring the various applications of data science in general. He actively contributes to open-source projects and continuously expands his knowledge through academic endeavors. Prabin actively participates in webinars, both nationally and internationally, as well as competing in diverse technology competitions. He also shares his expertise by conducting workshops on various technology topics. With a strong drive for excellence, Prabin aims to make valuable contributions in the field of AI while keeping up with the latest advancements.

PRABIN SHARMA POUDEL is an undergraduate senior at IOE's Thapathali Campus, has forever been in pursuit of knowledge and wisdom. His involvement extends beyond technical confines, with involvement in non-technical organizations like AYON. With a vision to become a scholarly figure, he aims to bridge multiple disciplines of knowledge. His current field of interest encompasses AI in conflict resolution, Computer vision, and AI in creative industries Beyond academics, Prabin finds joy in playing chess, composing music, and journaling his thoughts through poetry.

SANTOSH PANDEY is a fourth-year Bachelor's student at IOE, Thapathali Campus, with a passion for coding and technology. Currently working as an intern at NAAMII organization, he is driven by the goal of applying his skills to earn a living through coding. With a keen interest in full-stack development, graphic designing, mobile app development, and computer vision, Santosh is constantly exploring new technologies and honing his skills in these areas. In his free time, he actively participates in coding competitions and enjoys participating in marathons. Santosh's dedication, enthusiasm, and continuous learning make him a promising professional in the field of technology and coding.

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import os
dataset_path = "PCA Dataset"
# gets a list of image file names
image_files = os.listdir(dataset_path)
# orders the list of image file to make
them readable while printing
image_files.sort(key=lambda x:
int(x.split('(')[1].split(')')[0]))
print(image_files)

# Image 39 and 41 were taken separately,
so their size is different and needs
resizing.
from PIL import Image
import os
# Path to the folder containing the
images
dataset_path = 'PCA Dataset'
# List of images to resize
images_to_resize = ['image (39).jpg',
'image (41).jpg']
# Define the target size
target_size = (128, 128)

# Loop through the images and resize them
for image_name in images_to_resize:
    image_path =
os.path.join(dataset_path, image_name)
    image = Image.open(image_path)
    resized_image =
image.resize(target_size,
Image.ANTIALIAS)
    # Save the resized image back to the
same location
    resized_image.save(image_path)
print("Images resized successfully.")

from skimage import io
indices_to_print = [25, 26, 38, 44]
# Create a grid of subplots
fig, axes = plt.subplots(1,
len(indices_to_print), figsize=(15, 5))
```

```
# Load and display the selected images
for i, idx in
enumerate(indices_to_print):
    image_path =
os.path.join(dataset_path, f'image
({idx}).jpg')
    image = io.imread(image_path)
    axes[i].imshow(image)
    axes[i].set_title(f'Image {idx}')
    axes[i].axis('off')
plt.tight_layout()
extra_legend = "Roll:26,27,41"
plt.text(0.0001, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

# Print the 128 * 128 matrix for each
images
for image_name in image_files:
    image_path =
os.path.join(dataset_path, image_name)
    image = Image.open(image_path)
    # Convert the image to a numpy array
    image_array = np.array(image)
    print(f"Image Name: {image_name}")
    print("Image Pixel Values:")
    print(image_array)
    print("="*40)

# Lets pick the image number 25, and see
the matrix visualization and
concatenation process
image_path25 = 'PCA Dataset/image
(25).jpg'
image25 = Image.open(image_path25)
image25_array = np.array(image25)
print("Image Name: image (25).jpg")
print("Image Pixel Values:")
print(image25_array)

image25_array.shape
```

```
# Formation of row matrix of size = 1 *
49512. Done by Concatenation of all the
rows of image number 25
# Reshape the matrix to a row matrix
row_matrix25 = image25_array.reshape(1,-
1)
print("Row Matrix for image (25).jpg:")
print(row_matrix25)
row_matrix25.shape
# **Creation of Data_matrix of size = 44
* (16384 * 3). 44 rows mean the 44 images
and (16384 * 3) columns mean the pixel
values for each of those images. The
number 3 signifies for each RGB channel**
from PIL import Image
image_directory = 'PCA Dataset'
# List to store all row matrices
row_matrices = []

# Loop through all image files in the
directory
for image_file in
sorted(os.listdir(image_directory)):
    image_path =
os.path.join(image_directory, image_file)
    image = Image.open(image_path)
    image_array = np.array(image)
    row_matrix = image_array.reshape(1, -
1)
    row_matrices.append(row_matrix)

# Create the Data_matrix by stacking all
row matrices
Data_matrix = np.vstack(row_matrices)
# Print the shape of Data_matrix
print("Shape of Data_matrix:",
Data_matrix.shape)
# Print the Data_matrix
print("Data_matrix:")
print(Data_matrix)

# Converting to grayscale for
simplification
from skimage import io, color
dataset_path = "PCA Dataset"
output_path = "PCA Grayscale"
if not os.path.exists(output_path):
    os.makedirs(output_path)

# Loop through all images
for idx in range(1, 45):
    image_path =
os.path.join(dataset_path, f'image
({idx}).jpg')
    image = io.imread(image_path)
    grayscale_image =
color.rgb2gray(image)
    output_filename =
os.path.join(output_path,
f'grayscale_image_{idx}.jpg')
    io.imsave(output_filename,
grayscale_image)
    print(f'Saved grayscale image {idx}')

print("All images converted and saved as
grayscale.")
import os
dataset_path2 = "PCA Grayscale"
# gets a list of image file names
image_files2 = os.listdir(dataset_path2)
image_files2.sort(key=lambda x:
int(x.split('_')[2].split('.')[0]))
print(image_files2)
from skimage import io, color
indices_to_print = [25, 26, 38, 44]
# Create a grid of subplots
fig, axes = plt.subplots(1,
len(indices_to_print), figsize=(15, 5))

# Load and display the selected images
for i, idx in
enumerate(indices_to_print):
    image_path =
os.path.join(dataset_path, f'image
({idx}).jpg')
    image = io.imread(image_path)
    grayscale_image =
color.rgb2gray(image)
    axes[i].imshow(grayscale_image, cmap
= 'gray')
    axes[i].set_title(f'Image {idx}')
    axes[i].axis('off')
```

```
plt.tight_layout()
extra_legend = "Roll:26,27,41"
plt.text(0.0001, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

for image_name in image_files2:
    image_path =
os.path.join(dataset_path2, image_name)
    image = Image.open(image_path)
    # Convert the image to a numpy array
    image_array = np.array(image)
    print(f"Image Name: {image_name}")
    print("Image Pixel Values:")
    print(image_array)
    print("="*40)

gimage_path25 = 'PCA
Grayscale/grayscale_image_25.jpg'
gimage25 = Image.open(gimage_path25)
gimage25_array = np.array(gimage25)
print("Image Name:
grayscale_image_25.jpg")
print("Image Pixel Values:")
print(gimage25_array)
gimage25_array.shape

# **Formation of a row matrix for the
graysale image of image number 25**
# Reshape the matrix to a row matrix
grow_matrix25 =
gimage25_array.reshape(1,-1)
print("Row Matrix for
grayscale_image_25.jpg:")
print(grow_matrix25)
grow_matrix25.shape

# **Creation of Data_matrix of size = 44
* 16384. 44 rows mean the 44 images and
16384 columns mean the pixel values for
each of those images.**
```

```
from PIL import Image
import os
import numpy as np
dataset_path2 = 'PCA Grayscale'
# List to store all row matrices for
grayscale images
row_matrices2 = []
# Loop through all grayscale image files
in the directory
for image_file in
sorted(os.listdir(dataset_path2)):
    image_path =
os.path.join(dataset_path2, image_file)
    image = Image.open(image_path)
    image_array = np.array(image)
    row_matrix2 = image_array.reshape(1,
-1)
    row_matrices2.append(row_matrix2)

# Create the Data_matrix2 by stacking all
row matrices
Data_matrix2 = np.vstack(row_matrices2)
# Print the shape of Data_matrix2
print("Shape of Data_matrix2:",
Data_matrix2.shape)
# Print the Data_matrix2
print("Data_matrix2:")
print(Data_matrix2)

# **Calculation of Average_face_matrix.
By taking mean of all the columns(pixel
values across all images)**
# Calculate the average (mean) of each
column
average_face = np.mean(Data_matrix2,
axis=0)
# Create the average_face_matrix by
reshaping the average_face
average_face_matrix =
average_face.reshape(1, -1)
# Print the shape of average_face_matrix
print("Shape of average_face_matrix:",
average_face_matrix.shape)
# Print the average_face_matrix
print("average_face_matrix:")
print(average_face_matrix)
```

```
# **Resizing into 128 * 128 and printing
the image as the average_face of all
images**
from PIL import Image
# Reshape the average_face_matrix to
128x128
average_face_image =
average_face_matrix.reshape(128, 128)
# Convert the values to 8-bit unsigned
integer (0-255 range)
average_face_image = (average_face_image
).astype(np.uint8)
# Create a PIL Image from the array
average_face_pil =
Image.fromarray(average_face_image,
mode='L')
# Display the image
average_face_pil.show()

# Save the image
average_face_pil.save('average_face.jpg')
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

# Load the average_face_matrix image
average_face_image =
Image.open('average_face.jpg')

# Display the grayscale image
plt.figure(figsize=(4, 4))
plt.imshow(average_face_image,
cmap='gray')
plt.title('Average Face (Grayscale)')
plt.axis('off')
extra_legend = "Roll:26,27,41"
plt.text(0.345, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

# Convert to RGB and display
average_face_rgb =
average_face_image.convert('RGB')
plt.figure(figsize=(4, 4))
plt.imshow(average_face_rgb)
plt.title('Average Face (RGB)')
plt.axis('off')
extra_legend = "Roll:26,27,41"
plt.text(0.345, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

# **Normalize the Data_matrix2 by
subtracting from the average face**
# Calculate the normalized_face_matrix
normalized_face_matrix = Data_matrix2 -
average_face_matrix
# Print the shape of
normalized_face_matrix
print("Shape of normalized_face_matrix:",
normalized_face_matrix.shape)
# Print the normalized_face_matrix
print("normalized_face_matrix:")
print(normalized_face_matrix)

# **Covariance Matrix Calculation**
covariance_matrix =
np.cov(normalized_face_matrix, rowvar =
False)
print("Covariance matrix shape:",
covariance_matrix.shape)
plt.imshow(covariance_matrix,
cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.title('Covariance Matrix')
extra_legend = "Roll:26,27,41"
plt.text(0.345, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()
```

```
# **Eigen vectors and Eigen values
calculation**
eigenvalues, eigenvectors =
np.linalg.eig(covariance_matrix)
idx = np.argsort(eigenvalues)[::-1]  #
Reverse the order to sort in descending
order
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
eigenvectors.shape

k = 50  # Replace with the desired number
of eigenvectors
selected_eigenvectors = eigenvectors[:,
:k]
print(selected_eigenvectors.shape)
# Select the top 10 eigen vectors
(eigenfaces)
top_10_eigen_vectors =
selected_eigenvectors[:, :10]
top_10_transposed =
np.transpose(top_10_eigen_vectors)
# Create a subplot grid for displaying
the eigenfaces
fig, axs = plt.subplots(2, 5,
figsize=(12, 6))
fig.suptitle('Top 10 Eigenfaces')

# Iterate over the top 10 eigen vectors
(eigenfaces)
for i, ax in enumerate(axs.flat):
    eigenface_image =
np.real(top_10_transposed[i].reshape((128
, 128)))  # Assuming your image size is
128x128
    ax.imshow(eigenface_image,
cmap='gray')
    ax.set_title(f"Eigenface #{i+1}")
    ax.axis('off')

# Adjust the spacing between subplots
plt.tight_layout()
extra_legend = "Roll:26,27,41"
plt.text(0.345, 0.05, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
# Get the top 3 principal components
top_3_components =
selected_eigenvectors[:, :3]
# Project the data onto the top 3
components
projected_data =
np.dot(normalized_face_matrix,
top_3_components)
# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111,
projection='3d')
ax.set_title('Distribution of Faces in
Top 3 Principal Components')
# Plot each face as a point in 3D space
ax.scatter(projected_data[:, 0],
projected_data[:, 1], projected_data[:,
2], marker='x')

ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
plt.show()


# Proportion of Variance
# Calculate the proportion of variance
explained by each eigenvalue
pov_list = []
for i in range(0, k):  # Replace k with
the desired number of eigenvalues
    pov = eigenvalues[i] /
sum(eigenvalues)
    pov_list.append(pov)
```

```
# Calculate the cumulative explained
variance
cumulative_variance_list = []
total_variance = sum(eigenvalues)
cumulative_variance = 0
for i in range(0, k):
    cumulative_variance += eigenvalues[i]
    cumulative_variance_list.append(cumul
ative_variance / total_variance)

# Plot the proportion of variance and
cumulative explained variance
x = range(1, k+1)
plt.plot(x, pov_list, color="blue",
label='explained variance', alpha=1)
plt.plot(x, cumulative_variance_list,
color="green", label='cumulative
explained variance', alpha=0.65)
plt.grid()
# Set the axis labels
plt.xlabel('Eigenvalues')
plt.ylabel('Explained Variance')
plt.title('Proportion of Variance vs.
Eigenvalues')
# Add any extra legend information
extra_legend = "Roll:26,27,41"
plt.text(0.6, 0.1, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.legend()
plt.show()

print((pov_list))

import numpy as np
# Transpose mean data into row wise
transposed_normalized_face_matrix =
np.transpose(normalized_face_matrix)
print("shape of transposed
normalized_face_matrix
is",transposed_normalized_face_matrix.sha
pe)
# Transpose eigen vectors into row wise
for row feature vector
eigenvectors_transposed=np.transpose(eige
nvectors)
eigenvectors_transposed.shape
# **To project using any number of
desired Principal Components**
#for calculating final data for each
number of selected input vectors, loop
final_data={}
for j in range (1,71):
  print('If   %s  principal
components(eigenvectors) is/are
used'%(j))
  selected_eigenvectors=eigenvectors[:,
:j]
  row_zero_mean_data=
np.transpose(normalized_face_matrix)
  print('original dataset size =
',row_zero_mean_data.shape) #original
data= (60000 x 784), transposed= (784 X
60000)

  row_feature_vector=np.transpose(selecte
d_eigenvectors)
  print('shape of eigenvector matrix row-
wise =',row_feature_vector.shape)#eigen
vectors rowwise

  final_data[j] =
np.transpose(row_feature_vector @
row_zero_mean_data)
  print('final data shape
=',final_data[j].shape) # final data
plotted only in the top j principal
components discarding other dimensions
  print()

final_data[1]      # Projection using 1
PC
final_data[3]         # Projection using
3 PC
final_data[5]
final_data[7]        # using 7 PC
final_data[9]         # using 9 PC

np.real(final_data[57])
```

```
# Perform inverse transform and
reconstruct the data
def inverse_transform(transformed_data,
eigenvectors, mean):#mean of  original
data column-wise
    # Step 1: Multiply the transformed
data by the transpose of the eigenvectors
    # transformed data shape = (60000,k)
where k=1,2,3,4....784
    inverse_transformed_data =
np.dot(transformed_data, eigenvectors.T)
    # Step 2: Add the mean vector to the
result
    reconstructed_data =
inverse_transformed_data + mean
    return reconstructed_data

reconstructed_data={}
for no_of_pcs in range(2,70,2):
  print('WHEN RECONSTRUCTED FROM
%s  PRINCIPAL COMPONENTS(EIGENVECTORS)
'%(no_of_pcs-1))
  print()
  reconstructed_data[no_of_pcs-
1]=(inverse_transform(np.array(final_data
[no_of_pcs-1]),eigenvectors[:,
:no_of_pcs-1],average_face_matrix))
  print(inverse_transform(np.array(final_
data[no_of_pcs-1]),eigenvectors[:,
:no_of_pcs-1],average_face_matrix))
  print()

reconstructed_data1=np.real([reconstructe
d_data])
plt.imshow(np.real(reconstructed_data[61]
[17].reshape(128,128)),cmap='gray')
plt.imshow(np.real(reconstructed_data[7][
18].reshape(128,128)),cmap='gray')
num_pcs_values = [11, 21, 31, 41, 61]  #
Values for the number of principal
components
image_indices = [17, 18, 31, 38]  #
Indices of images

for img_idx in image_indices:
    for num_pcs in num_pcs_values:
        reconstructed_image =
np.real(reconstructed_data[num_pcs][img_i
dx].reshape(128, 128))
        plt.imshow(reconstructed_image,
cmap='gray')
        plt.title(f'Reconstructed Image
with {num_pcs} PCs, Image Index
{img_idx}')
        extra_legend = "Roll:26,27,41"
        plt.text(0.39, 0.05,
extra_legend, ha='left', va='center',
color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
        plt.legend()
        plt.show()

rd = np.real(reconstructed_data[41])
rd
dm = np.real(Data_matrix)
dm

from sklearn.metrics import
mean_squared_error
mse_values = []
for no_of_pcs_used in range(1, 51, 4):
    if no_of_pcs_used == 0:
        mse =
mean_squared_error(np.real(Data_matrix2),
np.real(reconstructed_data[1]))
        mse_values.append(mse)
    else:
        reconstructed_image =
reconstructed_data[no_of_pcs_used]
        mse =
mean_squared_error(np.real(Data_matrix2),
np.real(reconstructed_image))
        mse_values.append(mse)

print(mse_values)
no_of_components=[1,5,10,15,20,25,30,35,4
0,45,50,55,60]
plt.plot(no_of_components, mse_values,
marker='o')
```

```
plt.xlabel('Number of Principal
Components')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Reconstruction Quality: MSE
vs. Number of Principal Components')
extra_legend = "ROLL 26,27,41"
plt.text(0.4,0.9, extra_legend,
ha='left', va='center', color='red',
transform=plt.gca().transAxes,
         bbox=dict(facecolor='white',
edgecolor='red', boxstyle='round'))
plt.xticks(no_of_components)
plt.grid(True)
plt.show()
```