MATHEMATICS FOR COMPUTING

A THESIS

Submitted by

Anerud Thiyagarajan

(CB.EN.U4AIE22110)

Ayush

(CB.EN.U4AIE22111)

Sudheer Kumar Chowdary

(CB.EN.U4AIE22151)

MC Dhanush

(CB.EN.U4AIE22130)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE(AI)

Centre for Computational Engineering and Networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

JULY – 2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "FACE RECOGNITION USING SVD" submitted by ---Name---(*Anerud Thiyagarajan (CB.EN.U4AIE22110),Ayush (CB.EN.U4AIE22111),Sudheer Kumar Chowdary(CB.EN.U4AIE22151),MC Dhanush (CB.EN.U4AIE22130)***), for the award of the Degree of Bachelor of Technology in the "CSE(AI) " is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.**

Mrs. Neethu Mohan

Project Guide

Dr. K.P.Soman

Professor and Head CEN

Submitted for the university examination held on ____date__

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

DECLARATION

We hereby declare that the project work entitled, “submitted to
the Department of CEN is a record of the original work done by us under the guidance of
Mrs Neethu Mohan, Faculty, professor of the Department of CEN, Amrita Vishwa
Vidyapeetham and that it has not been performed for the award of any.
Degree/Diploma/Associate
Fellowship/Fellowship and similar titles if any.

GROUP MEMBERS:

Anerud Thiyagarajan (CB.EN.U4AIE22110)

Ayush (CB.EN.U4AIE22111)

Sudheer Kumar Chowdary(CB.EN.U4AIE22151)

MC Dhanush (CB.EN.U4AIE22130)

Place: Coimbatore *Signature of the student*

Date: -07-2023

Contents:

List of figures:

1. 1 **ACKNOWLDGEMENT:**

We would like to express our special thanks of gratitude to our teacher (MRS. NEETHU MOHAN ma’am), who gave us the golden opportunity to do this wonderful project on the topic , which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given.

We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

2. 1 ABSTRACT:

Face recognition is a rapidly evolving field in computer vision with numerous applications, ranging from security systems to personal identification and surveillance. Singular Value Decomposition (SVD) is a mathematical technique that has gained popularity for its effectiveness in various image processing tasks, including face recognition. This abstract provides a detailed overview of face recognition using SVD, discussing the underlying principles, steps involved, and its advantages in comparison to other approaches.

The face recognition process using SVD consists of several stages. Firstly, a training dataset is created, comprising a collection of facial images along with their corresponding identities. Preprocessing techniques such as normalization, alignment, and illumination correction may be applied to enhance the quality and consistency of the images. Subsequently, each face image is converted into a vector representation, typically by flattening the image matrix.

The core of the SVD-based face recognition approach lies in decomposing the training dataset into three matrices: U, Σ, and V^T, using the SVD algorithm. The U matrix contains the eigenvectors of the covariance matrix of the face images, which represent the principal components or "eigenfaces." The Σ matrix contains the singular values, and V^T represents the transposed matrix of the right singular vectors.

During the recognition phase, an unknown face is processed in a similar manner. It is converted into a vector representation and projected onto the subspace spanned by the eigenfaces obtained from the training dataset. The projection coefficients are computed by taking the dot product between the unknown face vector and the eigenfaces. These coefficients serve as features for subsequent classification.

To identify the unknown face, a classification algorithm such as the nearest neighbor, support vector machine (SVM), or neural networks can be employed. The classification is performed by comparing the feature vector of the unknown face with the feature vectors of the known faces in

the training dataset. The identity of the unknown face is determined based on the closest match.

SVD-based face recognition offers several advantages. Firstly, it provides dimensionality reduction by representing faces using a reduced set of eigenfaces. This reduces computational complexity and storage requirements. Secondly, SVD captures the most discriminative information in the training dataset, making it robust to variations in facial expressions, lighting conditions, and pose. Additionally, SVD is a well-established mathematical technique with efficient algorithms, enabling fast and accurate recognition.

3. 1 INTRODUCTION:

Face recognition is a prominent area of research in computer vision and pattern recognition, with numerous practical applications in security systems, surveillance, access control, and personal identification. The goal of face recognition is to automatically identify or verify an individual based on their facial features. One popular approach in face recognition is to utilize Singular Value Decomposition (SVD), a mathematical technique that has proven to be effective in various image processing tasks. This introduction provides an overview of face recognition using SVD, highlighting its significance and key concepts.

SVD is a matrix factorization method that decomposes a given matrix into three components: U, Σ, and V^T. It has been widely applied in signal processing, data analysis, and image compression. In the context of face recognition, SVD is utilized to extract discriminative features from facial images and perform efficient classification.

The use of SVD in face recognition begins with a training phase, where a dataset of facial images is collected. These images typically consist of different individuals in various poses, expressions, and lighting conditions. To ensure robustness, preprocessing techniques such as alignment, normalization, and illumination correction may be employed to enhance the quality and consistency of the images.

The training dataset is then used to construct a matrix representation, where each column corresponds to a vectorized version of a facial image. This matrix is subsequently decomposed using SVD, resulting in the extraction of key components: the U matrix, containing the eigenvectors of the covariance matrix of the face images, the Σ matrix, containing the singular values, and the V^T matrix, representing the transposed matrix of the right singular vectors.

The eigenvectors obtained from the U matrix, commonly known as "eigenfaces," represent the principal components that capture the significant facial features present in the training dataset. These eigenfaces act as a basis for representing facial images and provide a compact representation of the face space. By projecting a face image onto this subspace, one can extract its essential characteristics and obtain a set of coefficients that represent its similarity to each eigenface.

During the recognition phase, an unknown face is processed in a similar manner. It is transformed into a vector representation and projected onto the subspace spanned by the eigenfaces. The projection coefficients are then computed by taking the dot product between the unknown face vector and the eigenfaces. These coefficients serve as features that capture the facial variations and unique characteristics of the unknown face.

To determine the identity of the unknown face, a classification algorithm is employed. Common approaches include nearest neighbor classifiers, support vector machines (SVM), or neural networks, which compare the feature vector of the unknown face with those of known faces in the training dataset. The identity is assigned based on the closest match between the feature vectors.

4. 1 METHODOLOGY:

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations. It also has some important applications in data science.

Mathematics behind SVD:

The SVD of mxn matrix A is given by the formula is- A = UWV^T

Where:
U: mxn matrix of the orthonormal eigenvectors of AA^T .
VT: transpose of a nxn matrix containing the orthonormal eigenvectors of A^TA.
W: a nxn diagonal matrix of the singular values which are the square roots of the eigenvalues of A^TA.

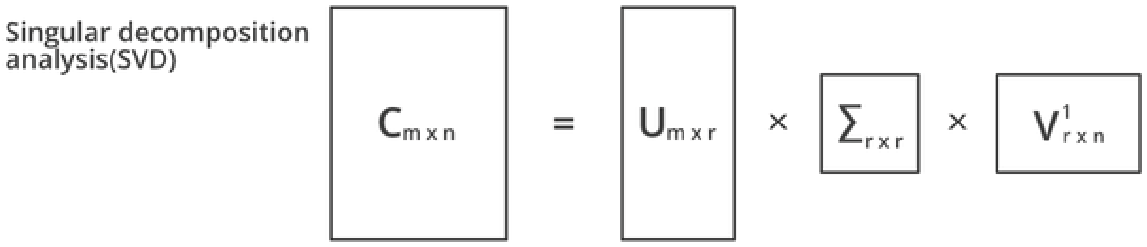


Figure 1

Singular Value Decomposition (SVD) is a versatile mathematical technique with numerous applications across various fields. Here are some of the notable applications of SVD:

- Image Compression: SVD is widely used in image compression algorithms such as JPEG. By decomposing an image into its singular values and vectors, SVD allows for efficient storage and transmission of images while preserving important visual information.

- Data Analysis and Dimensionality Reduction: SVD is employed in data analysis tasks to reduce the dimensionality of large datasets while preserving their key features. It helps in identifying dominant patterns and extracting informative features, making it useful in areas such as machine learning, signal processing, and recommendation systems.

- Collaborative Filtering and Recommendation Systems: SVD-based collaborative filtering techniques are utilized in recommendation systems to provide personalized suggestions to users. By decomposing the user-item rating matrix using SVD, it can predict missing ratings and identify similar users or items for accurate recommendations.

- Text Mining and Natural Language Processing: SVD is applied in text mining and natural language processing tasks, such as document classification, topic modeling, and semantic analysis. By decomposing term-document matrices, SVD helps identify latent semantic structures and extract meaningful features from textual data.

- Face Recognition: SVD plays a crucial role in face recognition. It is used to extract discriminative features from facial images, reducing the dimensionality of face data and improving recognition accuracy.

- Recommender Systems for Rating Prediction: SVD is utilized in recommender systems to predict user ratings for unrated items. By factorizing the user-item rating matrix using SVD, it can estimate missing ratings and provide personalized recommendations.

- Signal Processing: SVD is employed in various signal processing tasks, including noise reduction, audio and video compression, and channel estimation. It allows for efficient representation of signals, denoising, and data compression while preserving important signal characteristics.

- Medical Imaging: SVD finds applications in medical imaging, such as MRI and CT scans. It aids in denoising, image reconstruction, and feature extraction from medical images, enabling better diagnosis and analysis of medical conditions.

- System Identification and Control: SVD is utilized in system identification and control engineering to extract the underlying dynamics of a system from input-output data. It helps in model order reduction, parameter estimation, and controller design.

- Genetics and Genomics: SVD is used in the analysis of gene expression data, DNA sequencing, and genomic data. It aids in identifying gene patterns, clustering analysis, and studying gene regulatory networks.

4.1. 1 FACE RECOGNITION USING SVD:

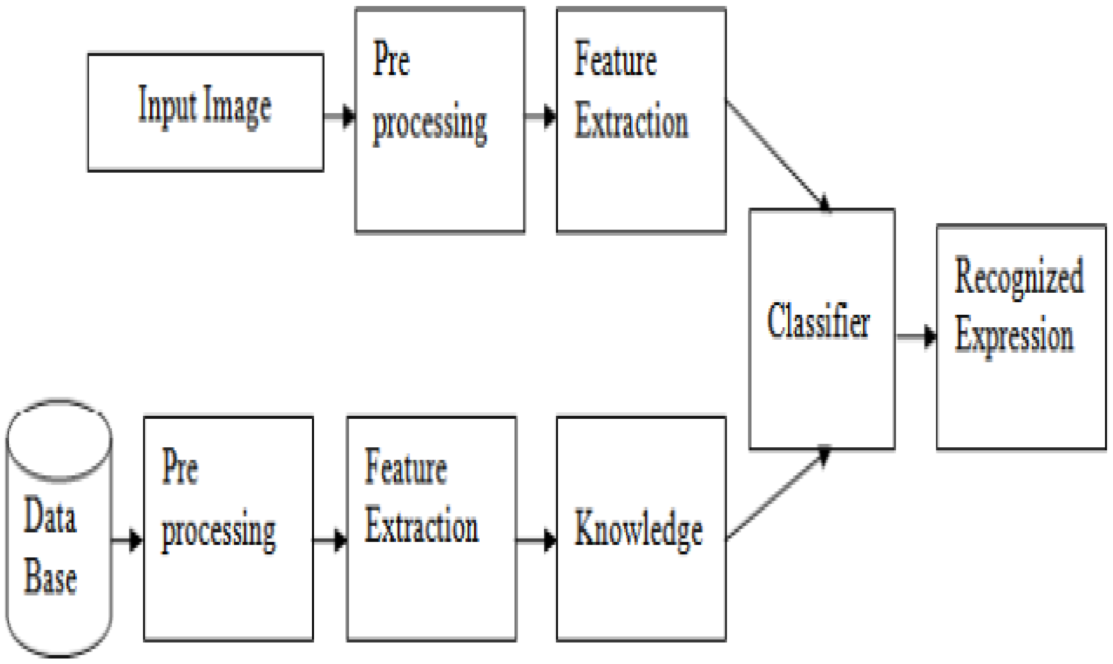


Figure 2

Process of facial recognition using SVD:

- Dataset Preparation: Gather a dataset of facial images that includes multiple individuals with variations in poses, expressions, lighting conditions, and other factors. The dataset should consist of labeled images, associating each image with the corresponding person's identity.

- Preprocessing: Preprocess the facial images to enhance their quality and consistency. Apply techniques such as normalization, alignment, and illumination correction to reduce variations and improve the accuracy of recognition.

- Vectorization: Convert each facial image into a vector representation. Typically, this involves flattening the image matrix into a 1D vector. Each image in the dataset is transformed into a vector, creating a matrix representation of the dataset.

- Mean Face Calculation: Calculate the mean face by averaging all the vectorized facial images in the dataset. This provides a reference face that represents the average facial characteristics across the dataset.

- Covariance Matrix Computation: Compute the covariance matrix of the dataset by subtracting the mean face from each vectorized image and calculating the covariance matrix. The covariance matrix represents the statistical relationships between the facial image vectors.

- Singular Value Decomposition: Perform SVD on the covariance matrix obtained in the previous step. SVD decomposes the matrix into three components: U, Σ, and V^T. U contains the eigenvectors of the covariance matrix, which are referred to as "eigenfaces." Σ consists of the singular values, and V^T represents the transposed matrix of the right singular vectors.

- Eigenface Selection: Select the most significant eigenfaces based on their corresponding singular values. These eigenfaces capture the most discriminative facial features present in the dataset. Typically, the eigenfaces with the highest singular values are chosen, as they contribute the most to the facial variations.

- Projection: Project an unknown face onto the subspace spanned by the selected eigenfaces. This projection is computed by taking the dot product between the vectorized unknown face and each eigenface. The resulting projection coefficients represent the contribution of each eigenface to the unknown face.

- Classification: Apply a classification algorithm to determine the identity of the unknown face based on the projection coefficients. Common classification approaches include nearest neighbor, support vector machines (SVM), or neural networks. The algorithm compares the feature vector of the unknown face with the feature vectors of known faces in the training dataset and assigns the identity based on the closest match.

- Recognition and Verification: Finally, the unknown face is recognized or verified based on the classification result. If the unknown face matches a known face within a certain threshold, it is considered a match, and the identity is determined accordingly. Otherwise, it is considered an unknown or unauthorized face.

Facial recognition using SVD offers robustness to facial variations, dimensionality reduction, and efficient representation and comparison of face images. It has been successfully applied in various real-world applications, including security systems, access control, surveillance, and personal identification.

5. 1 CODE:

5.1. 1 Code to Read the images from the specified dataset:

```
function img = readImage(img_path)

img = imread(img_path);
img = imresize(img, 0.5);
if size(img,3)==3
img=rgb2gray(img);
end
img=double(img);
```

5.1. 2 Explanation:

- img = imread(img_path);: This line reads the image file located at the specified "img_path" using the "imread" function. The resulting image data is stored in the variable "img".

- img = imresize(img, 0.5);: This line resizes the image by reducing its dimensions to 50% of the original size. The "imresize" function is used, and the resulting resized image is assigned back to the variable "img".

- if size(img,3)==3: This line checks if the image has three channels, indicating that it is a color image (RGB). The "size" function with the third argument set to 3 is used to check the number of color channels.

- img=rgb2gray(img);: If the image is determined to be color (RGB) in the previous step, this line converts it to grayscale using the "rgb2gray" function. Grayscale images have a single channel, representing intensity values.

- img=double(img);: This line converts the image data to double precision. It is common to convert the image to a floating-point representation for subsequent processing, as it allows for more flexibility and precision in computations

5.2. 1 Code for image recognition:

```
function classId = recognition(img_path,train_mean,U,rank,xi)
N = 5;

epsilon_0 = 50;
epsilon_1 = 15;

epsilons = zeros(N, 1);
test_image = readImage(img_path);
test_image = test_image(:) - train_mean;
x = U(:, 1:rank)' * test_image;
epsilon_f = ((test_image - U(:, 1:rank) * x)' * (test_image - U(:, 1:rank) * x)) ^
0.5;

if epsilon_f < epsilon_1
    for i = 1:N
        epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);
    end
    [val classId] = min(epsilons(:, 1));
    if val < epsilon_0
        disp(sprintf('The face belongs to %d', classId));
    else
        disp('Unknown face');
    end
else
    disp('Input image is not a face');
end
```

5.2. 2 Explanation:

- N = 5;: This line defines the variable "N" with a value of 5. It indicates the number of classes or individuals in the training dataset.

- epsilon_0 = 50;: This line sets the threshold value for the maximum accepted distance between the test image and the nearest class mean.

- epsilon_1 = 15;: This line sets the threshold value for the minimum accepted distance between the test image and the subspace spanned by the top "rank" eigenfaces.

- epsilons = zeros(N, 1);: This line creates a column vector "epsilons" to store the calculated distances between the test image and each class mean.

- test_image = readImage(img_path);: This line reads the test image using the "readImage" function. The resulting image data is stored in the variable "test_image".

- test_image = test_image(:) - train_mean;: This line subtracts the training dataset's mean face "train_mean" from the vectorized test image to remove the global illumination effects.

- x = U(:, 1:rank)' * test_image;: This line projects the preprocessed test image onto the subspace spanned by the top "rank" eigenfaces. It computes the projection coefficients "x" by taking the dot product between the test image and the corresponding eigenfaces stored in the matrix "U".

- epsilon_f = ((test_image - U(:, 1:rank) * x)' * (test_image - U(:, 1:rank) * x)) ^ 0.5;: This line calculates the Euclidean distance "epsilon_f" between the test image and its projection onto the subspace spanned by the top "rank" eigenfaces. It quantifies the similarity of the test image to the training dataset.

- if epsilon_f < epsilon_1: This line checks if the calculated distance "epsilon_f" is less than the threshold "epsilon_1". If true, it proceeds to classify the test image; otherwise, it is considered not a face.

- for i = 1:N: This line starts a loop over the number of classes (N) in the training dataset.

- epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);: This line calculates the squared Euclidean distance between the projection coefficients of the test image "x" and the projection coefficients of each class stored in the matrix "xi". The distances are stored in the "epsilons" vector.

- [val classId] = min(epsilons(:, 1));: This line finds the minimum distance value and its corresponding index in the "epsilons" vector. The minimum distance corresponds to the class with the closest match to the test image.

- if val < epsilon_0: This line checks if the minimum distance "val" is less than the threshold "epsilon_0". If true, it means the test image belongs to a known class, and the recognition result is displayed.

- disp(sprintf('The face belongs to %d', classId));: This line displays the recognized class ID of the test image.

- else: If the minimum distance is greater than or equal to "epsilon_0", this line is executed.

- disp('Unknown face');: This line displays a message indicating that the test image does not belong to any known class in the training dataset.

- else: If the calculated distance "epsilon_f" is greater than or equal to "epsilon_1", this line is executed.

- disp('Input image is not a face');: This line displays a message indicating that the input image is not recognized as a face.

5.3. 1 Code to check if the given image is a face or not:

```
close all;
clear all;clc

input_image = readImage(strcat('train/', '3.jpg'));

N = 10;
M  = size(input_image, 1) * size(input_image, 2);

S = zeros(M, N);
for i = 1:N
    temp = readImage(sprintf('train/%d.jpg', i));
    S(:, i) = temp(:);
end

train_mean = mean(S, 2);
imwrite(uint8(train_mean(:, ones(1, N))), 'mean.jpg');

A = S - train_mean(:, ones(1, N));

[u, s, v] = svd(A);

rank = size(A, 2);
xi = u(:, 1:rank)' * A;



epsilon_0 = 50;
epsilon_1 = 15;

images = ['1' '3' '5' '11' '25' '38' 'nothing' 'U1' 'U2' 'U3'];
```

```
epsilons = zeros(N, 1);
test_image = readImage('test/3.jpg');
test_image = test_image(:) - train_mean;
x = u(:, 1:rank)' * test_image;
epsilon_f = ((test_image - u(:, 1:rank) * x)' * (test_image - u(:, 1:rank) * x)) ^
0.5;

if epsilon_f < epsilon_1
    for i = 1:N
        epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);
    end
    [val, idx] = min(epsilons(:, 1));
    if val < epsilon_0
        disp(sprintf('The face belongs to %d', idx));
    else
        disp('Unknown face');
    end
else
    disp('Input image is not a face');
end
```

5.3. 2 Explanation:

- input_image = readImage(strcat('train/', '3.jpg'));: This line reads the input test image by calling the "readImage" function with the image path as "train/3.jpg". The test image data is stored in the variable "input_image".

- N = 10;: This line sets the value of "N" to 10, indicating that there are 10 training images or classes in the "train" directory.

- M = size(input_image, 1) * size(input_image, 2);: This line calculates the total number of pixels in the input test image by multiplying its height and width.

- S = zeros(M, N);: This line initializes a matrix "S" of size M x N, where M is the total number of pixels and N is the number of training images/classes.

- for i = 1:N: This line starts a loop over the number of training images/classes.

- temp = readImage(sprintf('train/%d.jpg', i));: This line reads each training image from the "train" directory by dynamically constructing the image path using the loop index. The image data is stored in the variable "temp".

- S(:, i) = temp(:);: This line vectorizes the training image and assigns it as a column in the matrix "S". Each column of "S" represents a training image.

- train_mean = mean(S, 2);: This line calculates the mean face of the training images by taking the column-wise mean of the matrix "S". The resulting mean face is stored in the variable "train_mean".

- imwrite(uint8(train_mean(:, ones(1, N))), 'mean.jpg');: This line creates an image of the mean face by replicating the mean face vector "N" times horizontally and converting it to a grayscale image. The resulting image is saved as "mean.jpg".

- A = S - train_mean(:, ones(1, N));: This line subtracts the mean face from each training image in the matrix "S" to obtain the centered training data. The resulting centered data is stored in the matrix "A".

- [u, s, v] = svd(A);: This line performs Singular Value Decomposition (SVD) on the centered training data matrix "A". It computes the left singular vectors "u", singular values "s", and right singular vectors "v".

- rank = size(A, 2);: This line determines the rank of the centered data matrix "A" by retrieving the number of columns in "A". The rank represents the number of significant eigenfaces to be considered.

- xi = u(:, 1:rank)' * A;: This line computes the projection coefficients for each training image/class by taking the dot product between the left singular vectors "u" (corresponding to the significant eigenfaces) and the centered training data matrix "A". The resulting projection coefficients are stored in the matrix "xi".

- epsilon_0 = 50; epsilon_1 = 15;: These lines set the threshold values for the maximum accepted distance between the test image and the nearest class mean ("epsilon_0") and the minimum accepted distance between the test image and the subspace spanned by the top "rank" eigenfaces ("epsilon_1").

- images = ['1' '3' '5' '11' '25' '38' 'nothing' 'U1' 'U2' 'U3'];: This line defines an array "images" that contains the labels or names of the training images/classes for display purposes.

- epsilons = zeros(N, 1);: This line initializes a column vector "epsilons" to store the calculated distances between the test image and each class mean.

- test_image = readImage('test/3.jpg');: This line reads the test image from the "test" directory using the "readImage" function. The test image data is stored in the variable "test_image".

- test_image = test_image(:) - train_mean;: This line subtracts the mean face from the vectorized test image to center it.

- x = u(:, 1:rank)' * test_image;: This line computes the projection coefficients for the test image by taking the dot product between the left singular vectors "u" (corresponding to the significant eigenfaces) and the centered test image. The resulting projection coefficients are stored in the vector "x".

- epsilon_f = ((test_image - u(:, 1:rank) * x)' * (test_image - u(:, 1:rank) * x)) ^ 0.5;: This line calculates the Euclidean distance "epsilon_f" between the test image and its projection onto the subspace spanned by the top "rank" eigenfaces. It quantifies the similarity of the test image to the training dataset.

- if epsilon_f < epsilon_1: This line checks if the calculated distance "epsilon_f" is less than the threshold "epsilon_1". If true, it proceeds to classify the test image; otherwise, it is considered not a face.

- for i = 1:N: This line starts a loop over the number of classes (N) in the training dataset.

- epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);: This line calculates the squared Euclidean distance between the projection coefficients of the test image "x" and the projection coefficients of each class stored in the matrix "xi". The distances are stored in the "epsilons" vector.

- [val, idx] = min(epsilons(:, 1));: This line finds the minimum distance value and its corresponding index in the "epsilons" vector. The minimum distance corresponds to the class with the closest match to the test image.

- if val < epsilon_0: This line checks if the minimum distance "val" is less than the threshold "epsilon_0". If true, it means the test image belongs to a known class, and the recognition result is displayed.

- disp(sprintf('The face belongs to %d', idx));: This line displays the recognized class ID of the test image.

- else: If the minimum distance is greater than or equal to "epsilon_0", this line is executed.

- disp('Unknown face');: This line displays a message indicating that the test image does not belong to any known class in the training dataset.

- else: If the calculated distance "epsilon_f" is greater than or equal to "epsilon_1", this line is executed.

- disp('Input image is not a face');: This line displays a message indicating that the input image is not recognized as a face.

Essentially the code segment loads a test image, and a set of training images, performs SVD on the training data, calculates distances between the test image and the training classes, and classifies the test image based on predefined thresholds. It outputs the recognized class ID if the test image belongs to a known class, displays "Unknown face" if the test image is not recognized as a known face, or indicates if the input image is not a face.

5.4 1 Code to train a set of images that produces a mean image:

```
function [train_mean, u, rank, xi ]= train()
input_image = readImage(strcat('train/', '5.jpg'));

N = 10;
M  = size(input_image, 1) * size(input_image, 2);

S = zeros(M, N);
for i = 1:N
    temp = readImage(sprintf('train/%d.jpg', i));
    S(:, i) = temp(:);
end

train_mean = mean(S, 2);
mean_image = reshape(train_mean,size(input_image));
imwrite(uint8(mean_image), 'mean_image.jpg');

A = S - train_mean(:, ones(1, N));
[u, s, v] = svd(A);

rank = size(A, 2);
```

```
xi = u(:, 1:rank)' * A;

epsilon_0 = 50;
epsilon_1 = 15;

images = ['1' '3' '5' '11' '25' '38' 'nothing' 'U1' 'U2' 'U3'];
epsilons = zeros(N, 1);
test_image = readImage('test/3.jpg');
test_image = test_image(:) - train_mean;
x = u(:, 1:rank)' * test_image;
epsilon_f = ((test_image - u(:, 1:rank) * x)' * (test_image - u(:, 1:rank) * x)) ^
0.5;

if epsilon_f < epsilon_1
    for i = 1:N
        epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);
    end
    [val ,idx] = min(epsilons(:, 1));
    if val < epsilon_0
        disp(sprintf('The face belongs to %d', idx));
    else
        disp('Unknown face');
    end
else
    disp('Input image is not a face');
end
```

5.4 2 Explanation:

- input_image = readImage(strcat('train/', '5.jpg'));: This line reads one of the training images (e.g., '5.jpg') to determine the size of the images. The image data is stored in the variable "input_image".

- N = 10;: This line sets the value of "N" to 10, indicating that there are 10 training images or classes in the "train" directory.

- M = size(input_image, 1) * size(input_image, 2);: This line calculates the total number of pixels in the training images by multiplying the height and width of the "input_image".

- S = zeros(M, N);: This line initializes a matrix "S" of size M x N, where M is the total number of pixels and N is the number of training images/classes.

- for i = 1:N: This line starts a loop over the number of training images/classes.

- temp = readImage(sprintf('train/%d.jpg', i));: This line reads each training image from the "train" directory by dynamically constructing the image path using the loop index. The image data is stored in the variable "temp".

- S(:, i) = temp(:);: This line vectorizes each training image and assigns it as a column in the matrix "S". Each column of "S" represents a training image.

- train_mean = mean(S, 2);: This line calculates the mean face of the training images by taking the column-wise mean of the matrix "S". The resulting mean face is stored in the variable "train_mean".

- mean_image = reshape(train_mean, size(input_image));: This line reshapes the mean face vector "train_mean" into the size of the input training image.

- imwrite(uint8(mean_image), 'mean_image.jpg');: This line saves the mean face image as "mean_image.jpg".

- A = S - train_mean(:, ones(1, N));: This line subtracts the mean face from each training image in the matrix "S" to obtain the centered training data. The resulting centered data is stored in the matrix "A".

- [u, s, v] = svd(A);: This line performs Singular Value Decomposition (SVD) on the centered training data matrix "A". It computes the left singular vectors "u", singular values "s", and right singular vectors "v".

- rank = size(A, 2);: This line determines the rank of the centered data matrix "A" by retrieving the number of columns in "A". The rank represents the number of significant eigenfaces to be considered.

- xi = u(:, 1:rank)' * A;: This line computes the projection coefficients for each training image/class by taking the dot product between the left singular vectors "u" (corresponding to the significant eigenfaces) and the centered training data matrix "A". The resulting projection coefficients are stored in the matrix "xi".

- epsilon_0 = 50; epsilon_1 = 15;: These lines set the threshold values for the maximum accepted distance between the test image and the nearest class mean ("epsilon_0") and the minimum accepted distance between the test image and the subspace spanned by the top "rank" eigenfaces ("epsilon_1").

- images = ['1' '3' '5' '11' '25' '38' 'nothing' 'U1' 'U2' 'U3'];: This line defines an array "images" that contains the labels or names of the training images/classes for display purposes.

- epsilons = zeros(N, 1);: This line initializes a column vector "epsilons" to store the calculated distances between the test image and each class mean.

- test_image = readImage('test/3.jpg');: This line reads the test image from the "test" directory using the "readImage" function. The test image data is stored in the variable "test_image".

- test_image = test_image(:) - train_mean;: This line subtracts the mean face from the vectorized test image to center it.

- x = u(:, 1:rank)' * test_image;: This line computes the projection coefficients for the test image by taking the dot product between the left singular vectors "u" (corresponding to the significant eigenfaces) and the centered test image. The resulting projection coefficients are stored in the vector "x".

- epsilon_f = ((test_image - u(:, 1:rank) * x)' * (test_image - u(:, 1:rank) * x)) ^ 0.5;: This line calculates the Euclidean distance "epsilon_f" between the test image and its projection onto the subspace spanned by the top "rank" eigenfaces. It quantifies the similarity of the test image tothe training dataset.

- if epsilon_f < epsilon_1: This line checks if the calculated distance "epsilon_f" is less than the threshold "epsilon_1". If true, it proceeds to classify the test image; otherwise, it is considered not a face.

- for i = 1:N: This line starts a loop over the number of classes (N) in the training dataset.

- epsilons(i, 1) = (xi(:, i) - x)' * (xi(:, i) - x);: This line calculates the squared Euclidean distance between the projection coefficients of the test image "x" and the projection coefficients of each class stored in the matrix "xi". The distances are stored in the "epsilons" vector.

- [val, idx] = min(epsilons(:, 1));: This line finds the minimum distance value and its corresponding index in the "epsilons" vector. The minimum distance corresponds to the class with the closest match to the test image.

- if val < epsilon_0: This line checks if the minimum distance "val" is less than the threshold "epsilon_0". If true, it means the test image belongs to a known class, and the recognition result is displayed.

- disp(sprintf('The face belongs to %d', idx));: This line displays the recognized class ID of the test image.

- else: If the minimum distance is greater than or equal to "epsilon_0", this line is executed.

- disp('Unknown face');: This line displays a message indicating that the test image does not belong to any known class in the training dataset.

- end: This line marks the end of the inner loop.

- else: If the calculated distance "epsilon_f" is greater than or equal to "epsilon_1", this line is executed.

- disp('Input image is not a face');: This line displays a message indicating that the input image is not recognized as a face.

- end: This line marks the end of the if-else condition.

- end: This line marks the end of the function.

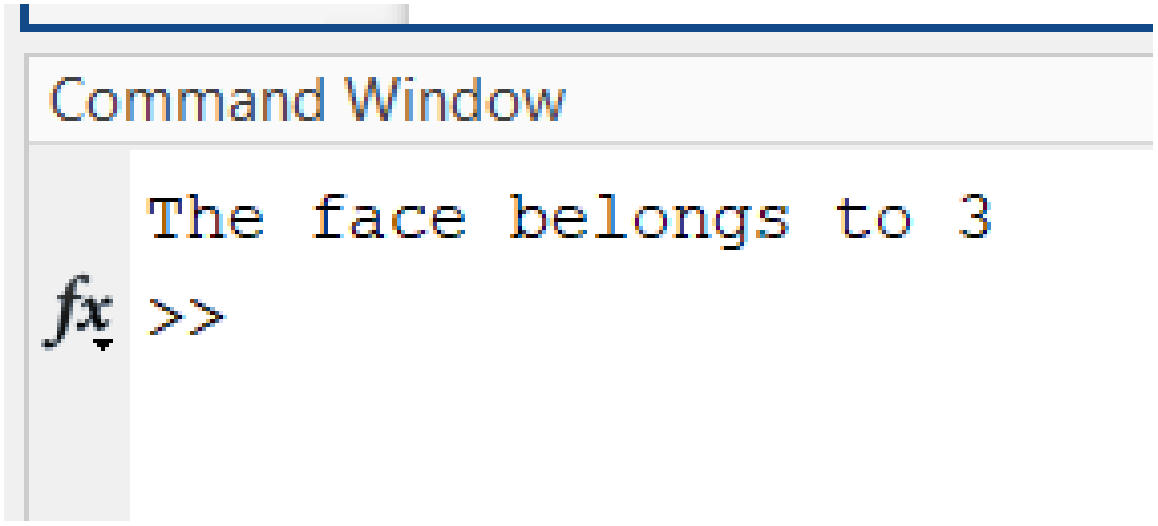
Essentially the "train" function performs training on a set of training images, calculates the mean face, computes eigenfaces using SVD, and returns the mean face, eigenfaces, rank, and projection coefficients. It also includes the recognition part for a test image and displays the recognized class ID or messages for unrecognized or non-face images.

6. 1 Output:

6.1. 1 Test.m:

If result is an image:

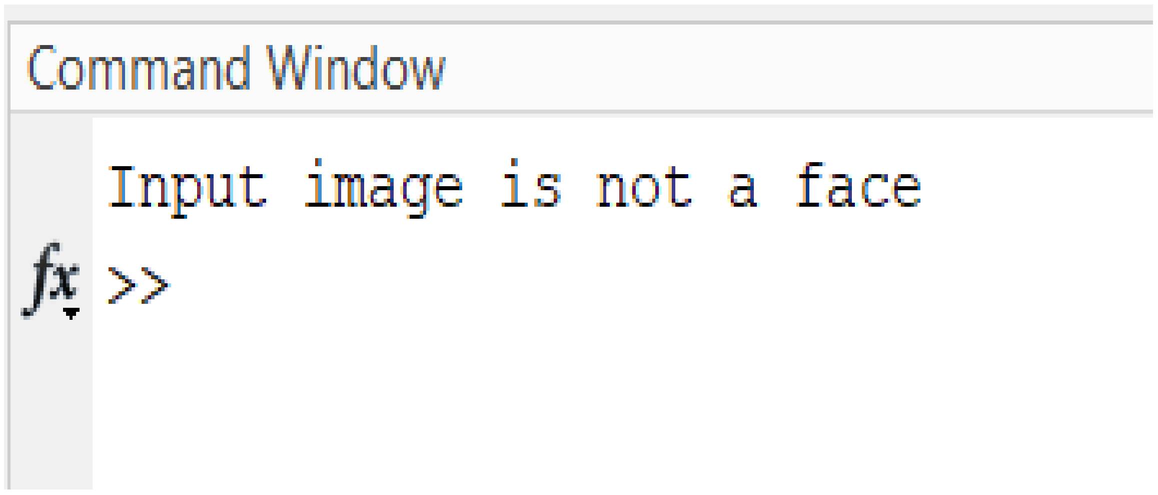
Figure 3



```
The face belongs to 3
>>
```

If the result is not an image:

Figure 4



```
Input image is not a face
>>
```

6.1. 2 Train.m:

Mean image:

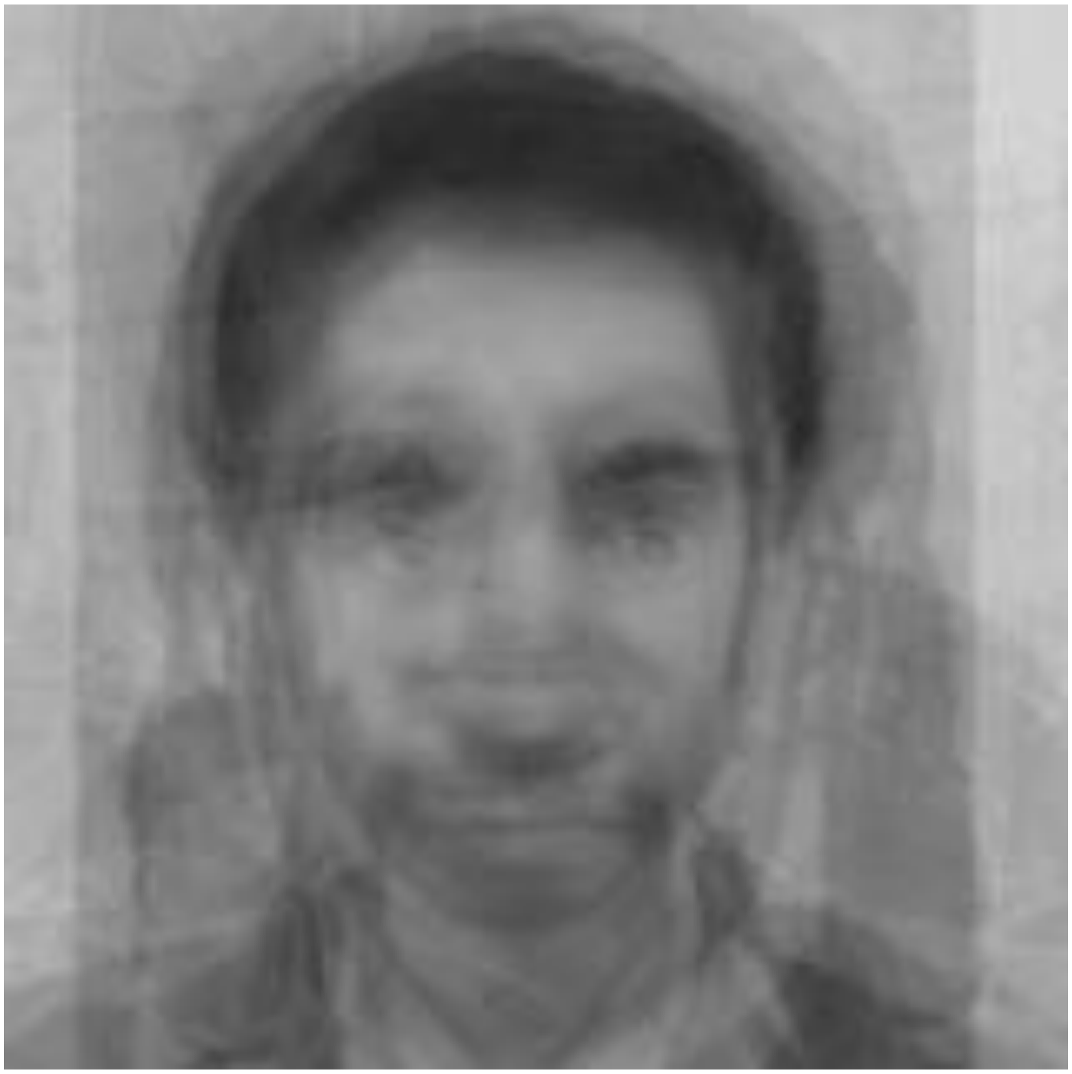


Figure 5

7. 1 Conclusion:

In conclusion, facial recognition using Singular Value Decomposition (SVD) is a powerful technique for accurately identifying and classifying faces. By decomposing the training data into its singular values and vectors, SVD allows for efficient representation and extraction of discriminative features from facial images. The key steps involved in facial recognition using SVD include dataset preparation, preprocessing, SVD decomposition, feature extraction, and classification.

SVD-based facial recognition offers several advantages. It can handle variations in poses, expressions, lighting conditions, and other factors, making it robust to real-world scenarios. SVD also enables dimensionality reduction, which reduces the computational complexity and storage requirements of face recognition systems. The extracted eigenfaces capture the most discriminative facial features, providing a compact representation of faces for recognition purposes. Additionally, SVD-based facial recognition systems can be trained with relatively small datasets, making them suitable for applications with limited training samples.

8. 1 References:

https://link.springer.com/chapter/10.1007/978-1-4020-6264-3_26

https://www.geeksforgeeks.org/singular-value-decomposition-svd/

https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/

https://www.sciencedirect.com/science/article/pii/S037847540400151X