Ahmedabad University

Facial Recognition using Eigenfaces

Problem: Implement the Eigenfaces algorithm for facial recognition using Principal Component Analysis (PCA). Multivariate Calculus Concepts: Eigenvalue decomposition, matrix calculus.

CS Application: PCA in facial recognition is used to reduce the dimensionality of the data and extract key features, relying on matrix operations that involve multivariate calculus.

Darshil Shah, Varun Kotwani, Shrimad Patel

EigenFaces Introduction

Eigenfaces is a method used for facial recognition based on Principal Component Analysis (PCA), which is grounded in several key mathematical concepts like calculus

Key Concepts

1. Principal Component Analysis (PCA)
    a. Data Representation
    b. Mean Centering
    c. Covariance Matrix
    d. Eigenvectors and Eigenvalues
    e. Selecting Principal Components
2. Reconstruction of Images
3. Recognition (Classification)
    a. Project the Test Image
    b. Compare with Training Data

CODE

```python
import cv2
import os
import numpy as np

# Set a common size for all images (e.g., 200x200 pixels)
IMAGE_WIDTH = 200
IMAGE_HEIGHT = 200
CONFIDENCE_THRESHOLD = 9000  # Adjust this value based on experimentation

# Function to read images and labels from the dataset
def prepare_training_data(data_folder_path):
    dirs = os.listdir(data_folder_path)
    faces = []
    labels = []
    label_names = []

    for dir_name in dirs:
        if not dir_name.startswith("."):
            label_name = dir_name  # Use the folder name as the label
            label_names.append(label_name)
            subject_dir_path = os.path.join(data_folder_path, dir_name)
            subject_images_names = os.listdir(subject_dir_path)

            for image_name in subject_images_names:
                image_path = os.path.join(subject_dir_path, image_name)
                image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

                if image is None:
                    continue

                # Resize the image to a fixed size
                resized_image = cv2.resize(image, (IMAGE_WIDTH, IMAGE_HEIGHT))

                faces.append(resized_image)
                labels.append(label_name)

    return faces, labels, label_names
```

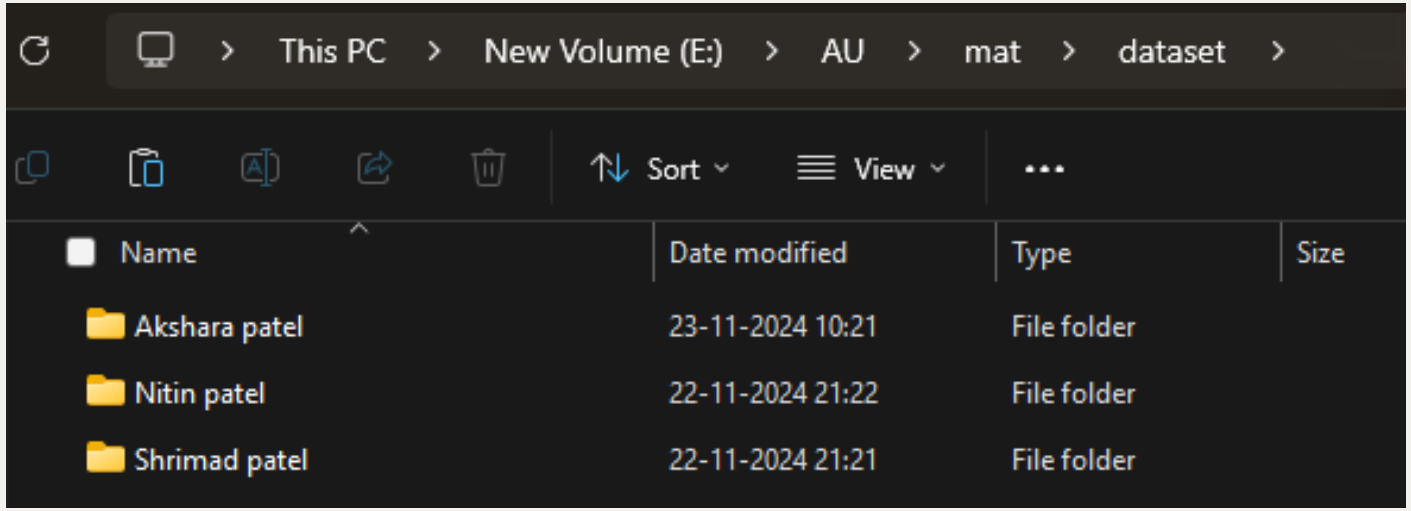
CODE

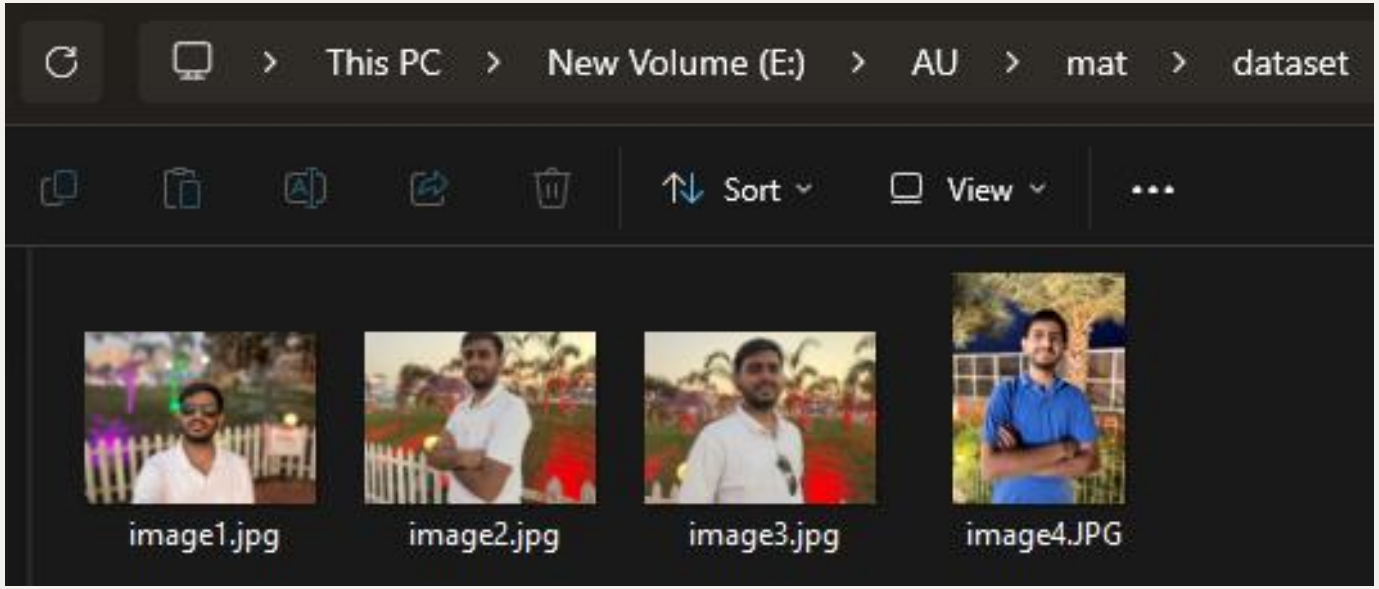
```python

# Function to recognize faces in a test image
def predict(test_img, face_recognizer, subjects):
    img = test_img.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5)

    for (x, y, w, h) in faces:
        roi_gray = gray[y:y+w, x:x+h]
        roi_gray_resized = cv2.resize(roi_gray, (IMAGE_WIDTH, IMAGE_HEIGHT))  # Resize to match training size

        # Predict the label and confidence score
        label_index, confidence = face_recognizer.predict(roi_gray_resized)

        print(f"Confidence score: {confidence}")  # Print the confidence score for debugging

        if confidence < CONFIDENCE_THRESHOLD:
            label_name = subjects[label_index]
        else:
            label_name = "Unknown"

        # Draw rectangle and label name on the image
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(img, f"{label_name} ({confidence:.2f})", (x, y-5), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    return img
```

CODE

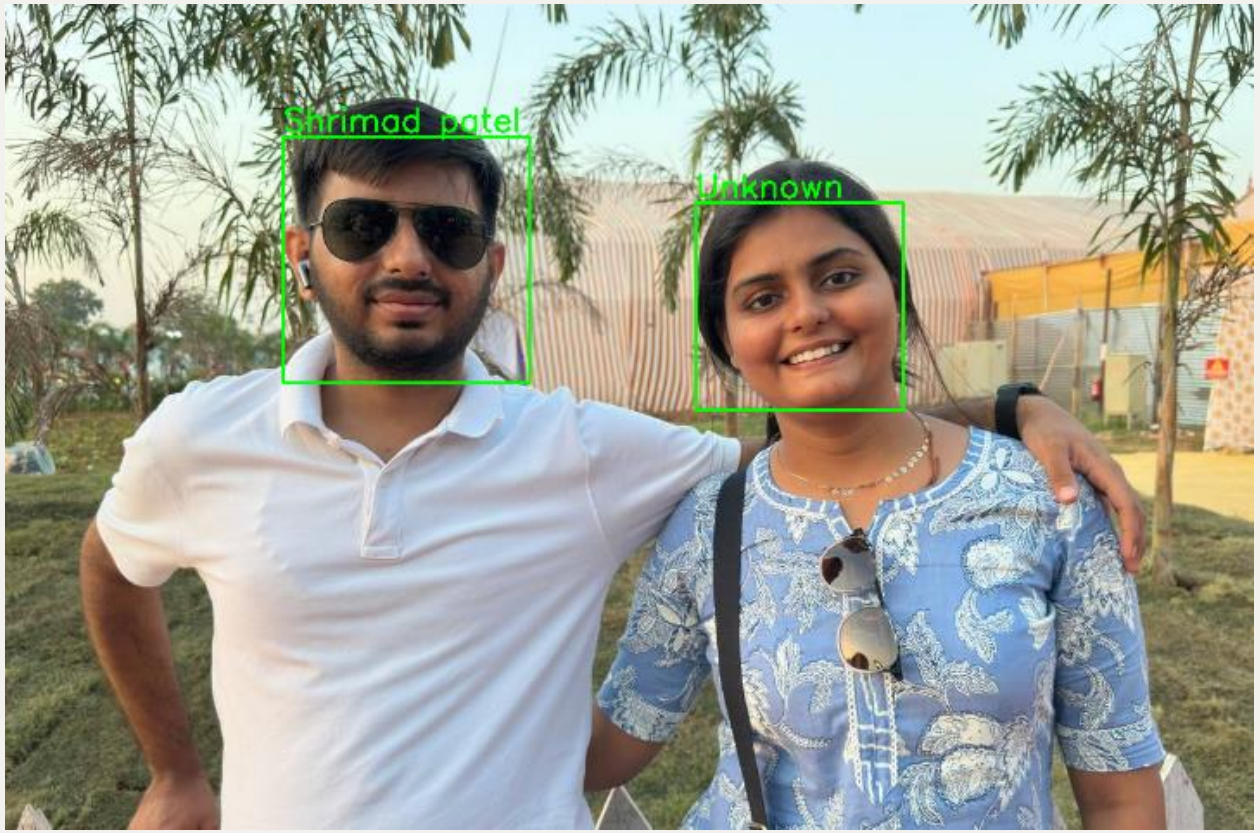
```
66
67  # Step 1: Prepare training data
68  print("Preparing data...")
69  faces, labels, label_names = prepare_training_data("dataset")
70  print("Data prepared.")
71
72  # Convert names to numerical labels for training
73  unique_labels = list(set(labels))
74  label_to_index = {name: index for index, name in enumerate(unique_labels)}
75  indexed_labels = [label_to_index[name] for name in labels]
76
77  # Step 2: Train the EigenFace Recognizer
78  face_recognizer = cv2.face.EigenFaceRecognizer_create()
79
80  print("Training model...")
81  face_recognizer.train(faces, np.array(indexed_labels))
82  print("Training completed.")
83
84  # Step 3: Test the trained model
85  # Prepare the label list for prediction
86  subjects = unique_labels  # This list will map numerical indices to names
87
88  # Load a test image
89  test_image_path = r"E:\Photos\bakrol-vadtal trip\5870771173653071636.jpg"  # Use raw string for backslashes
90  test_image = cv2.imread(test_image_path)
91
92  predicted_img = predict(test_image, face_recognizer, subjects)
93
94  # Display the result
95  cv2.imshow("Prediction", predicted_img)
96  cv2.waitKey(0)
97  cv2.destroyAllWindows()
```

Dataset Image

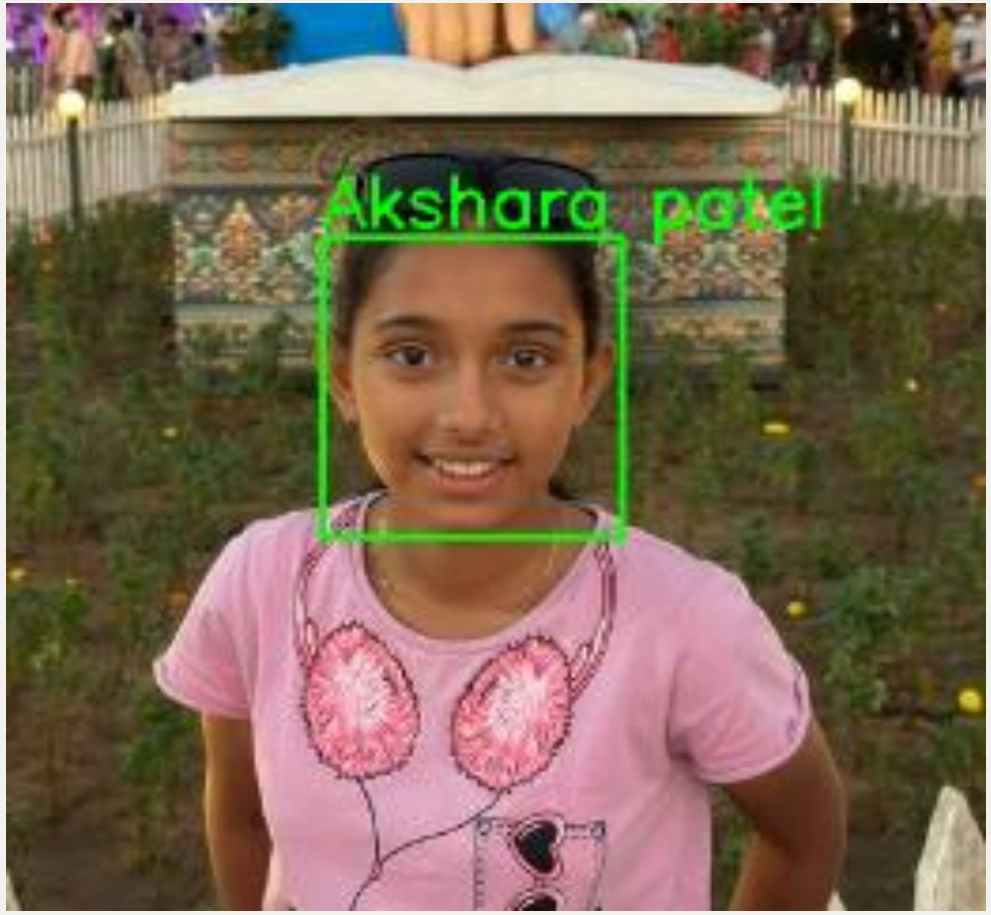


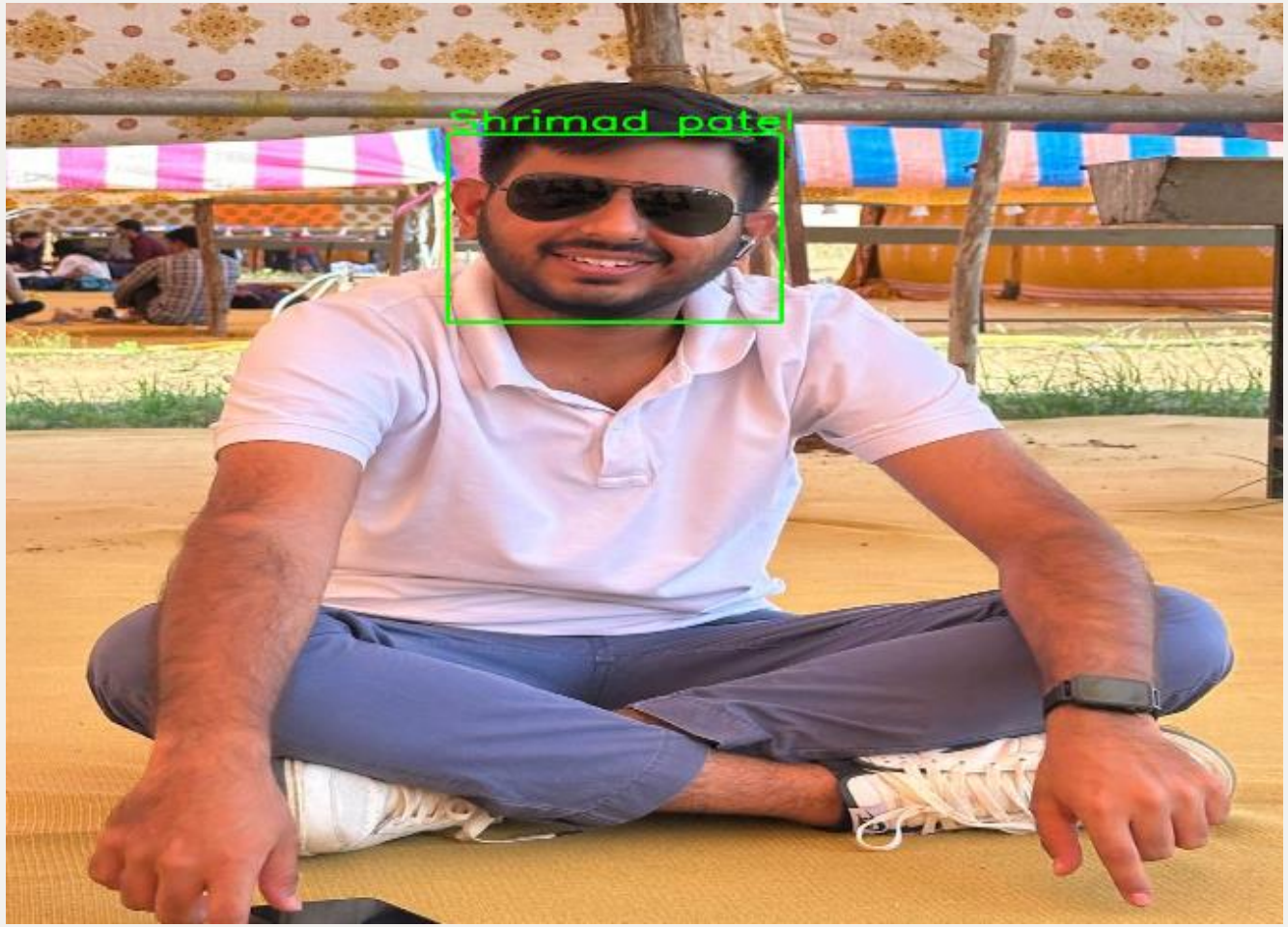


Results



Results





Ahmedabad University

Flow of the Project

- First we read all the training images.
- Convert each [m x n] image matrix into [(m x n) x 1] image vector.
- Find average-face-vector, and subtract it from every image vector.
- Stack all in matrix forming A = [(m x n) x i] matrix (where i = number of images).
- Calculate covariance matrix of above matrix
- Find eigenvectors and eigenvalues of above covariance matrix.
- Choose best k eigenvectors
- Find weights of each image and store it

Testing Algorithm

- Normalize the test image -> I = Test-Image-Vector — Average-Face-Vector.
- Find weights of test image using above training proces
- Calculate error between weights of test image and weights of all images in training set.
- Choose the image of training set which has minimum error.

Thank You