

# **“FACE RECOGNITION”**

**A**  
**Project Report**  
*Submitted by*

**AKSHAT BHATNAGAR**

**22BCON737**

*In partial fulfilment for the award of the degree*

*Of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE & ENGINEERING**

**At**

**JECRC UNIVERSITY, JAIPUR**

**December 2024**



## **ACKNOWLEDGEMENT**

Many people have supported me, in different ways, during the work with the thesis. I'd like to thank my guide Mr Sahil Chhabra and Mr Sandeep Gautam & HOD Dr Shashi Sharma for their kind and active support and valuable guidance during the work process. My family has as always offered me their unconditional support, thank you! I have taken efforts in the Project. However, it would not have been possible without the kind support and many individuals and organizations. I would like to extend my sincere thanks to each and every members related to JECRC University.

Student Name- AKSHAT BHATNAGAR

Registration No.- 22BCON737

## Candidate's Declaration

I, AKSHAT BHATNAGAR, bearing roll number 22BCON737, hereby declare that the work which is being presented in the Project, entitled “**FACE RECOGNITION**” in partial fulfilment for award of Degree of “**Bachelor of Technology**” in Department of **Computer Science Engineering** is submitted to the Department Computer Science & Engineering, JECRC University is a record of Project work carried under the Guidance of MR SAHIL and MR SANDEEP sir, Department of Computer Science & Engineering.

I have not submitted the matter presented in this work anywhere for the award of any other Degree.

Student Name-AKSHAT BHATNAGAR

Computer Science Engineering with data science and data analytics

Enrolment No.: 22BCON737

## **JECRC UNIVERSITY JAIPUR**

Ramchandrapura, Sitapura Industrial Area Extn., Jaipur-303905(Raj.) India  
www.jecrcuniversity.edu.in

Date: 10-05-2025

### **CERTIFICATE**

Certified that the Project Report entitled “**FACE RECOGNITION**” submitted by AKSHAT BHATNAGAR bearing roll no. 22BCON737 in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology at JECRC University, Jaipur is a record of the student's own work carried out under my supervision and guidance. To the best of my knowledge, this Project work has not been submitted to JECRC University or any other university for the award of the degree. It is further understood that by this certificate the undersigned does not endorse or approve of any statement made, opinion expressed or conclusion drawn therein but approve Project for the purpose for which it is submitted.

**Guide Name- MR SAHIL CHHABRA**

(Project Guide)

**Dr. SHASHI SHARMA**

(Project Coordinator)

(Dy. HOD, CSE)

## **ABSTRACT**

This project presents a real-time face recognition system developed using Python and open-source libraries such as face\_recognition, OpenCV, and NumPy. The system is capable of detecting and recognizing human faces captured through a webcam by comparing them against a dataset of known individuals. It employs machine learning and computer vision techniques, particularly the use of deep learning models via the face\_recognition library, which generates 128-dimensional encodings of faces. These encodings are matched using distance metrics to identify or label unknown individuals. The project includes modules for face data collection, model training using PCA and SVM, and real-time recognition. Use cases include automated attendance systems, face-based login, surveillance, and access control. Designed with modularity and scalability in mind, this system serves as both an educational tool and a prototype for practical biometric applications, demonstrating how AI can be integrated into daily-use identity verification systems.

## **List of Tables**

1. Test Case Results – Black Box Testing
  2. Test Case Results – White Box Testing
  3. Hardware Requirements – Minimum vs. Recommended
  4. Software Requirements and Tools
  5. Comparison of Face Recognition Techniques (e.g., Eigenfaces vs Fisherfaces vs Deep Learning Models)
  6. Summary of Key Deep Learning Models
- 

## **List of Figures**

1. System Architecture Diagram
  2. Data Flow Diagram (DFD – Level 1)
  3. UML Use Case Diagram
  4. UML Class Diagram
  5. UML Sequence Diagram
  6. UML Collaboration Diagram
  7. UML Object Diagram
  8. Screenshot – Known Face Detected
  9. Screenshot – Unknown Face Detected
  10. Screenshot – Initial Webcam Feed
-

## List of Symbols

1.  $\rightarrow$  (Arrow) – Used in flowcharts and sequence diagrams
  2.  $\square$  (Rectangle) – Represents process blocks or class elements
  3.  $\diamond$  (Diamond) – Conditional decision nodes in DFD/UML
  4.  $\{ \}$  – Sets or data structures in pseudocode
  5.  $\parallel$  or  $||$  – Parallel processing or comments in code
- 

## List of Abbreviations

1. AI – Artificial Intelligence
2. CNN – Convolutional Neural Network
3. DFD – Data Flow Diagram
4. GUI – Graphical User Interface
5. HOG – Histogram of Oriented Gradients
6. IDE – Integrated Development Environment
7. LDA – Linear Discriminant Analysis
8. LBP – Local Binary Pattern
9. ML – Machine Learning
10. PCA – Principal Component Analysis
11. SVM – Support Vector Machine
12. SRS – Software Requirement Specification
13. XML – eXtensible Markup Language

- 14. API – Application Programming Interface (mentioned indirectly via libraries)
- 15. KNN – K-Nearest Neighbors
- 16. OS – Operating System
- 17. RAM – Random Access Memory



TABLE OF CONTENTS	Page
DECLARATION .....	2
CERTIFICATE.....	3
ACKNOWLEDGEMENTS .....	4
ABSTRACT.....	5
LIST OF TABLES.....	6
LIST OF FIGURES.....	7
LIST OF SYMBOLS .....	8
LIST OF ABBREVIATIONS.....	9
INTRODUCTION.....	11
1.1 AUTOMATED ATTENDANCE SYSTEMS.....	11
1.2 FACE-BASED LOGIN / AUTHENTICATION.....	12
1.3 SURVEILLANCE AND SECURITY MONITORING.....	13
1.4 SMART ACCESS CONTROL SYSTEMS.....	14
1.5 PERSONALIZED USER EXPERIENCES.....	15
1.6 IDENTITY VERIFICATION IN SENSITIVE .....	16
1.7 FACEDATACOLLECTION.....	17
1.8 FACE RECOGNITION MODULE.....	18
1.9 PRETRAINED HAAR CASCADE FACE DETECTOR.....	19
1.10 KNN ALGO.....	20
LITERATURE SURVEY.....	21
2.1 TRADITIONAL APPROACHES.....	24
2.2 MACHINE LEARNING-BASED METHODS.....	27
3.2 DEEP LEARNING AND CNNs.....	29
2.4 REAL-TIME FACE RECOGNITION.....	31
SOFTWARE REQUIREMENT SPECIFICATION.....	33
3.1. INTRODUCTION.....	33
3.2.FUNCTIONAL REQUIREMENTS.....	34
3.3. NON-FUNCTIONAL REQUIREMENTS.....	35

3.4. SOFTWARE REQUIREMENTS.....	36
3.5. HARDWARE REQUIREMENTS.....	36
SOFTWARE DESIGN.....	37
DFDs.....	37
UML.....	38
CLASS DIAGRAM.....	38
INTERACTION DIAGRAM SEQUENCE.....	39
DIAGRAM.....	40
OBJECT DIAGRAM.....	40
USECASE DAIGRAM.....	41
CONTROL FLOW DIAGRAM.....	42
DATABASE DIAGRAM.....	43
E-R DIAGRAM.....	44
SOFTWARE AND HARDWARE DIAGRAM.....	45
CODE TEMPLATES.....	47
TESTING.....	50
BLACK BOX.....	52
WHITE BOX.....	53
OUTPUT SCREEN.....	54
CONCLUSION.....	57
RECOMMENDATION.....	58
BIBLIOGRAPHY.....	60
APPENDICES.....	62

# CHAPTER 1

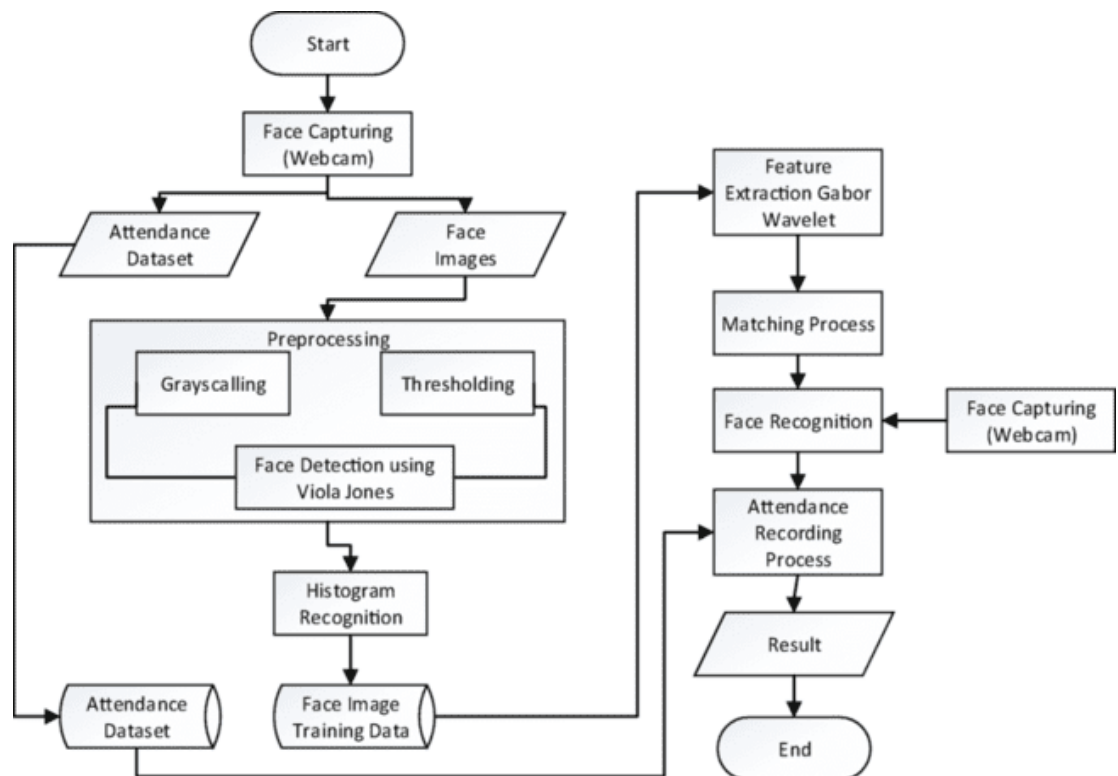
## INTRODUCTION

The Face Recognition is a Python-based application that demonstrates how to implement a basic but effective facial recognition system using open-source tools. It focuses on recognizing and labeling human faces in real-time video streams by leveraging machine learning, computer vision, and deep learning techniques.

Use cases of this project are

### 1.1 Automated Attendance Systems

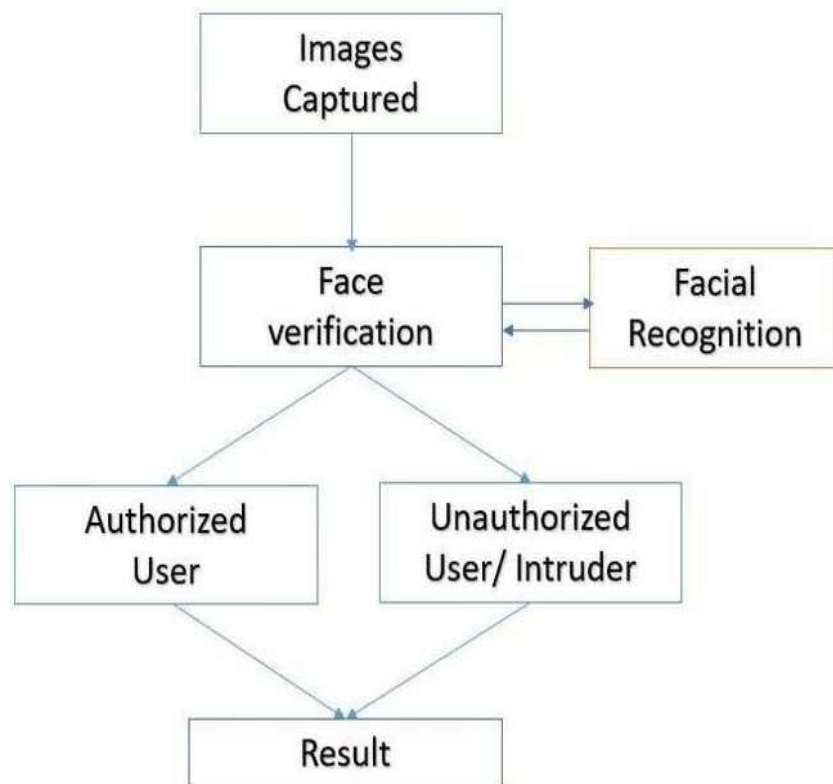
- **Description:** Recognize and log students or employees as "present" when their face is detected by a camera.
- **Use Case:** Schools, colleges, corporate offices.
- **Benefits:** No manual entry needed; time-efficient and tamper-proof.



### Automated Attendance Systems

## 📱 1.2 Face-Based Login / Authentication

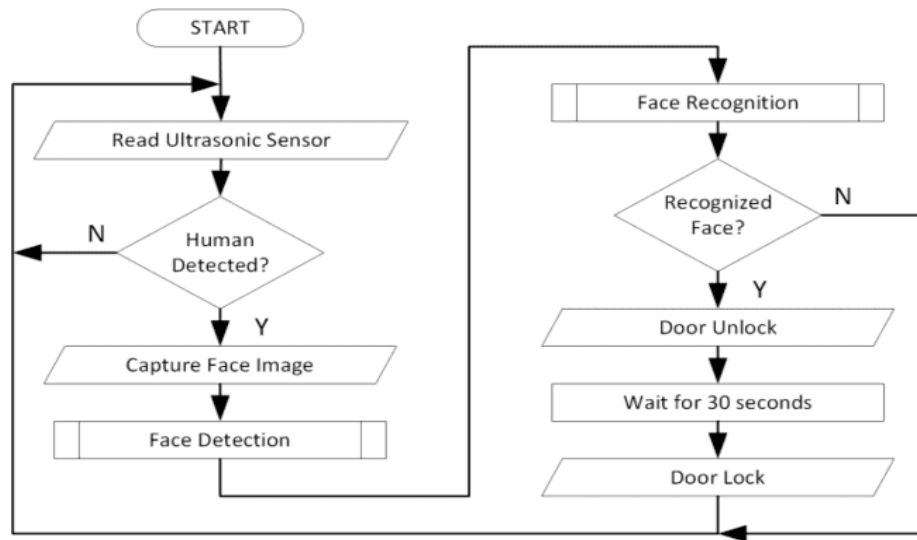
- **Description:** Use facial recognition instead of traditional passwords to log into devices or secure systems.
- **Use Case:** Personal computers, mobile apps, restricted software access.
- **Benefits:** Contactless, secure, and convenient.



### Face-Based Login / Authentication

## 📺 1.3 Surveillance and Security Monitoring

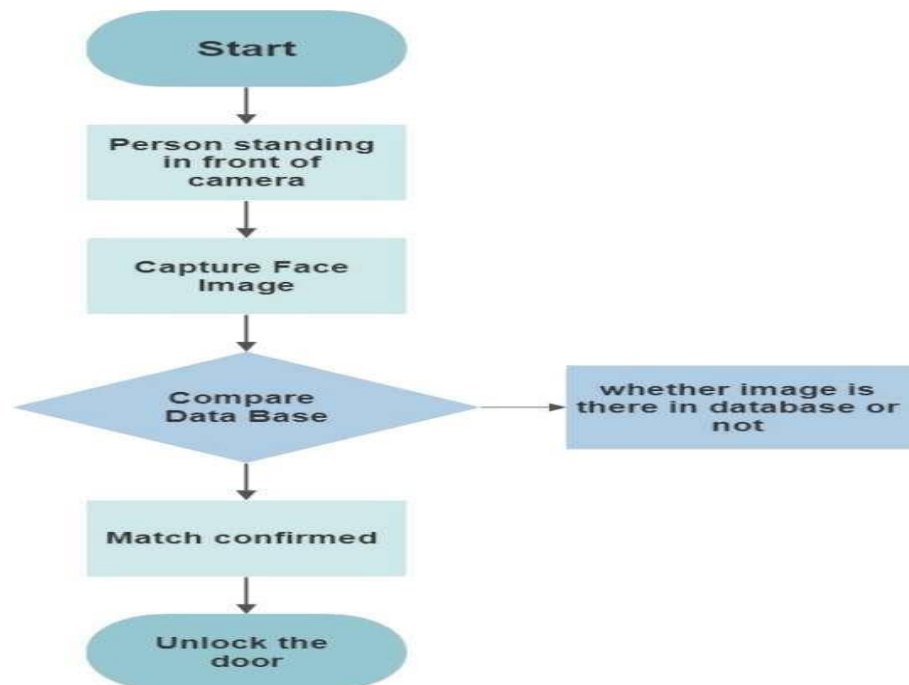
- **Description:** Recognize known individuals or detect intruders in real-time via CCTV or webcam feeds.
- **Use Case:** Airports, railway stations, smart homes, government buildings.
- **Benefits:** Enhanced public safety, automated alerts, faster response time.



### Surveillance and Security Monitoring

#### 1.4 Smart Access Control Systems

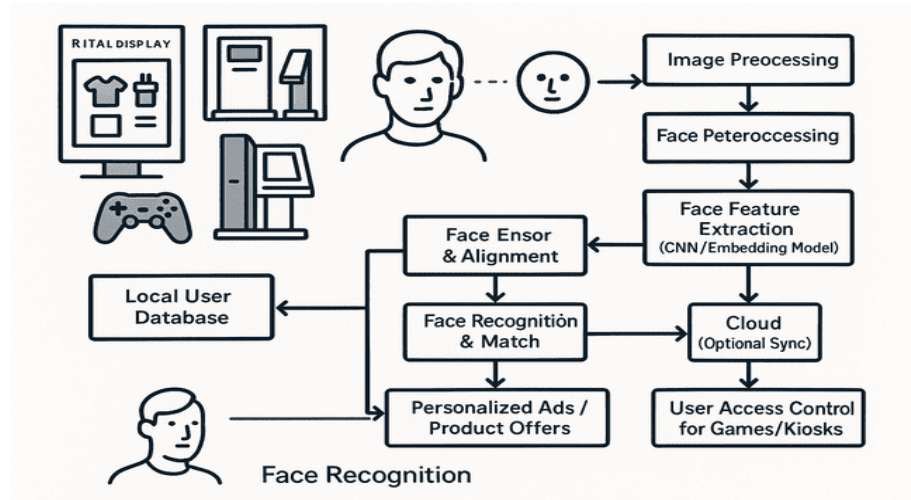
- **Description:** Grant or deny access to physical locations (e.g., doors, gates) based on face recognition.
- **Use Case:** Smart homes, offices, locker rooms, bank vaults.
- **Benefits:** Replaces keys/cards; ensures access only to authorized personnel.



### Smart Access Control Systems

## 🔗 1.5. Personalized User Experiences

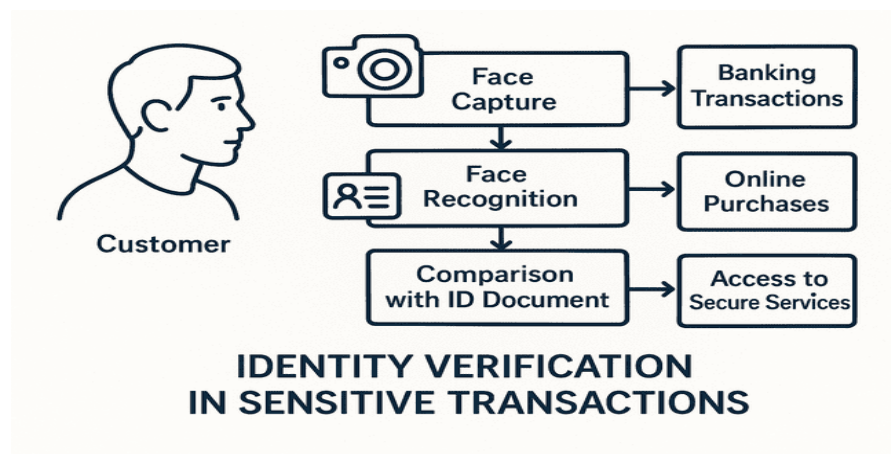
- **Description:** Detect and recognize users to offer customized content or settings.
- **Use Case:** Retail displays, gaming consoles, kiosks, advertising panels.
- **Benefits:** Increases engagement, delivers targeted content.



### Personalized User Experiences

## 🔗 1.6 Identity Verification in Sensitive Transactions

- **Description:** Match live face with government ID photos to verify identity.
- **Use Case:** e-KYC, online banking, telemedicine, remote hiring.
- **Benefits:** Reduces fraud, ensures compliance with regulations.



### Identity Verification in Sensitive Transactions

The primary aim of the project is to identify individuals by comparing facial features against a known dataset. This is achieved using the `face_recognition` library, a high-level Python module built on top of `dlib`, which provides deep learning models capable of detecting facial landmarks and generating unique facial encodings. These encodings serve as a numerical representation of a face and are used to match new faces against previously stored data.

In this implementation:

- The system loads sample images of known individuals and extracts their facial encodings.
- It then activates the system's webcam using `OpenCV` to capture video frames in real time.
- For every frame, it detects faces and computes their encodings.
- These are compared with the known faces using a simple distance metric, and the names of recognized individuals are displayed on the screen.
- Unknown faces are labeled accordingly, allowing for dynamic and flexible recognition.

This project is particularly useful as an educational tool and a prototype for real-world applications such as:

- Automated attendance systems
- Access control systems
- Security surveillance
- Human-computer interaction interfaces

By integrating `OpenCV` and `face_recognition`, the project offers a lightweight and efficient solution to the face recognition problem, with minimal code and infrastructure requirements. The design is modular and can be easily scaled or improved with more complex techniques like deep neural networks or real-time alerting.

deeper understanding of the role and working of each file within the overall face recognition pipeline.

## 1.1 face\_data\_collection.ipynb – Face Data Collection Module

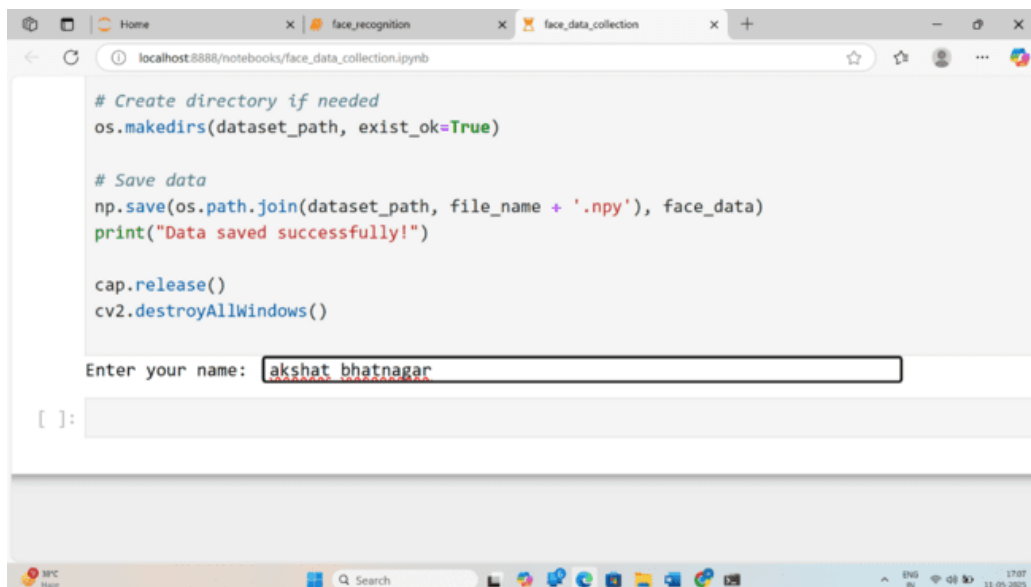
This Jupyter Notebook is the first step in the face recognition pipeline. It is responsible for capturing and storing face data of individuals, which will later be used to train the recognition system.

### How it works:

- The script uses OpenCV to access the device's webcam.
- A Haar Cascade Classifier (haarcascade\_frontalface\_alt.xml) is loaded to detect faces in each frame.
- As the camera captures video, it detects the face region in real time and extracts it.
- The extracted face is converted to grayscale, resized to a fixed dimension (typically 100x100 pixels), and stored as an image.
- Each image is saved with a filename format like person.<user\_id>.<image\_number>.jpg, making it easy to identify which face belongs to which user.

### Purpose:

- This script helps in building a labeled dataset, which is essential for training the face recognition model.
- Multiple face images per person (usually 50–100) ensure that the model learns a robust representation of each face.



```
# Create directory if needed
os.makedirs(dataset_path, exist_ok=True)

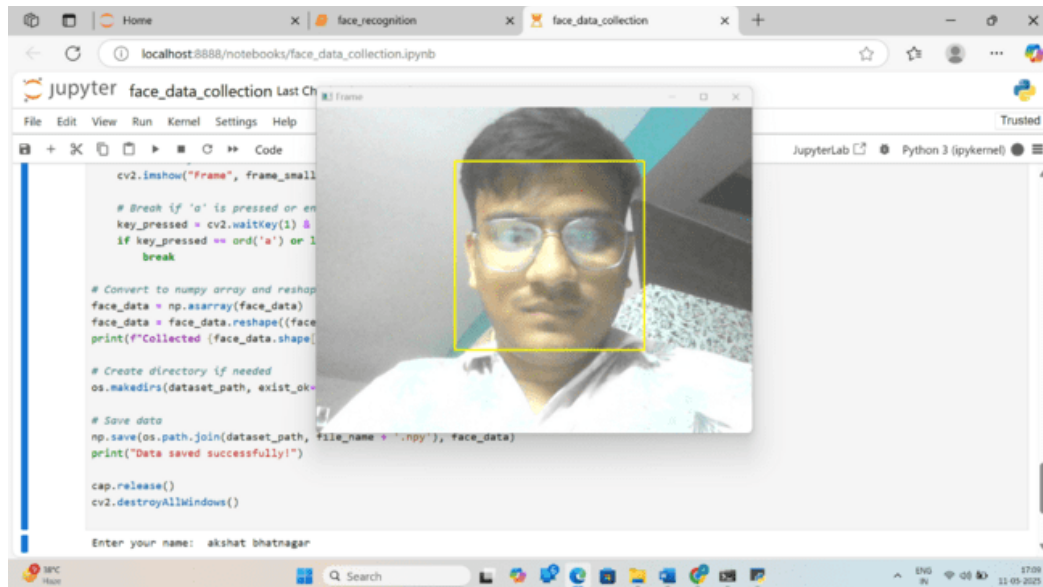
# Save data
np.save(os.path.join(dataset_path, file_name + '.npy'), face_data)
print("Data saved successfully!")

cap.release()
cv2.destroyAllWindows()
```

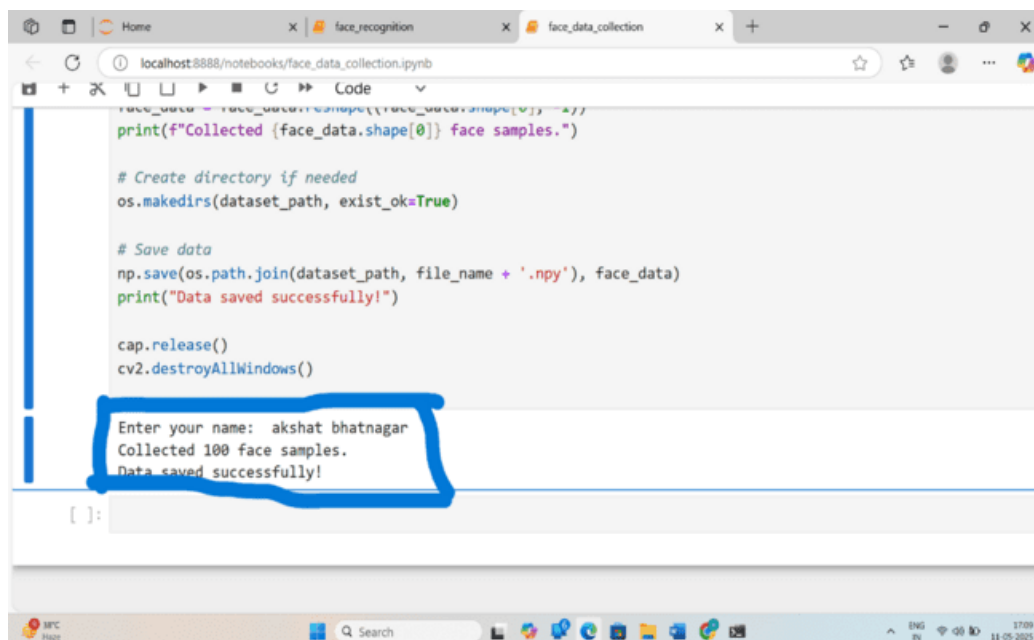
Enter your name:



## Face Data Collection Module



## Face data collection



## Face data collected successfully

### 1.2. face\_recognition.ipynb – Face Recognition Module

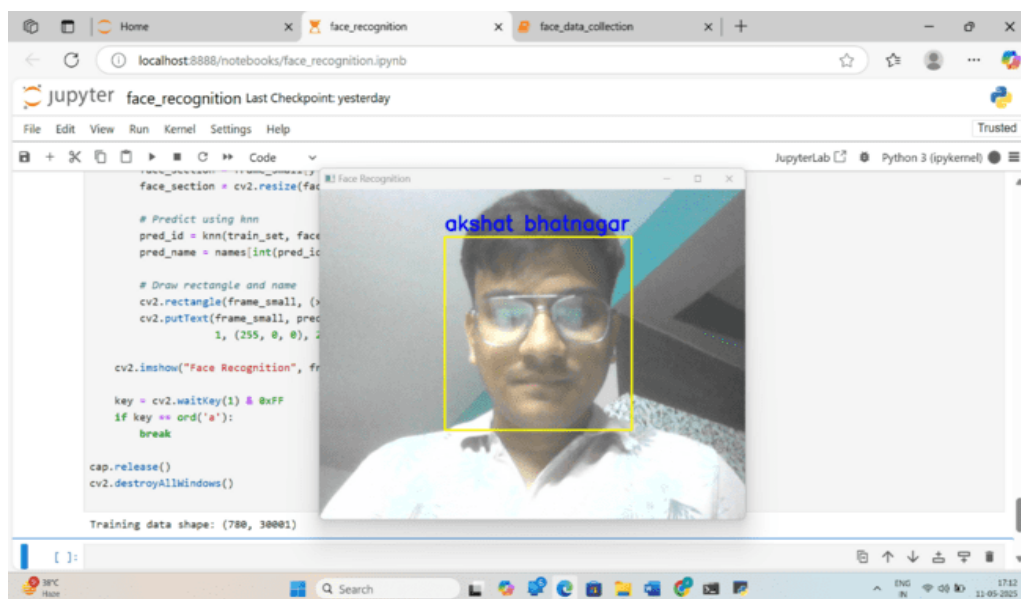
This notebook is the core part of the project that performs face recognition in real time using the webcam.

## 🔍 How it works:

- The script loads the previously saved face images (from face\_data\_collection.ipynb) and flattens each image into a 1D array to prepare a training dataset.
- It applies Principal Component Analysis (PCA) for dimensionality reduction. PCA reduces the number of features while preserving the important patterns in the image data — this improves model efficiency and reduces overfitting.
- After dimensionality reduction, it trains a Support Vector Machine (SVM) classifier. SVM is a powerful supervised learning algorithm suitable for classification tasks like face recognition.
- The webcam then captures live frames, and faces are detected using the Haar Cascade again.
- Detected face regions are preprocessed (grayscale, resize) and passed through the same PCA transformation and SVM model to predict the identity.
- If the prediction confidence is high, the person's name (or ID) is displayed on the screen with a bounding box. If the confidence is low or the face is not in the dataset, it is labeled as "Unknown."

## 🤖 Purpose:

- Implements an end-to-end pipeline for real-time face recognition.
- Demonstrates classical machine learning (PCA + SVM) for image classification.



## Face recognition

### 1.3. haarcascade\_frontalface\_alt.xml – Pretrained Haar Cascade Face Detector

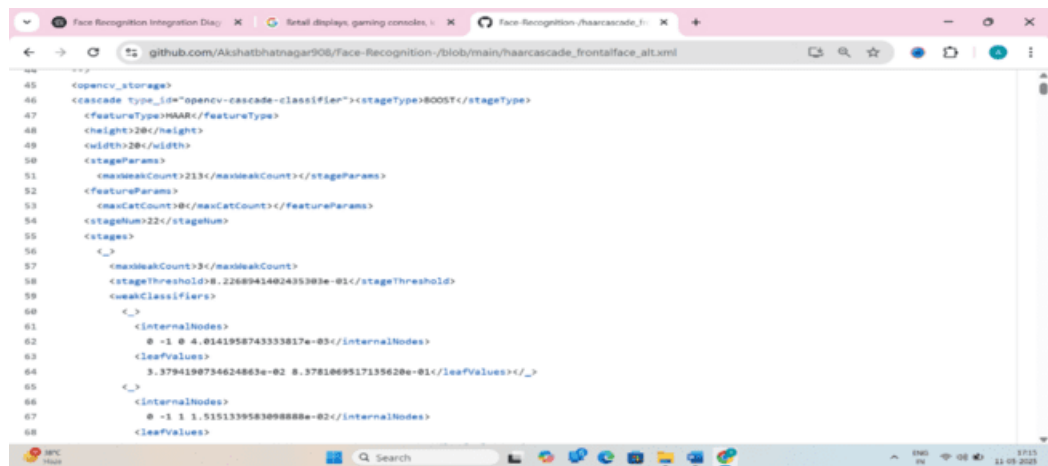
This is an XML configuration file that contains the parameters for a Haar Cascade Classifier, which is a machine learning-based approach for object detection developed by Viola and Jones.

#### How it works:

- It contains trained data for detecting frontal human faces.
- When used in OpenCV's CascadeClassifier, it scans the image at multiple scales to find patterns that match the characteristics of a human face.
- It is efficient and suitable for real-time applications because it uses a cascade of increasingly complex classifiers.

#### Purpose in this project:

- Used in both the data collection and recognition notebooks to detect face regions before processing.
- Serves as the foundation for capturing and focusing on face areas, rather than processing the entire image.

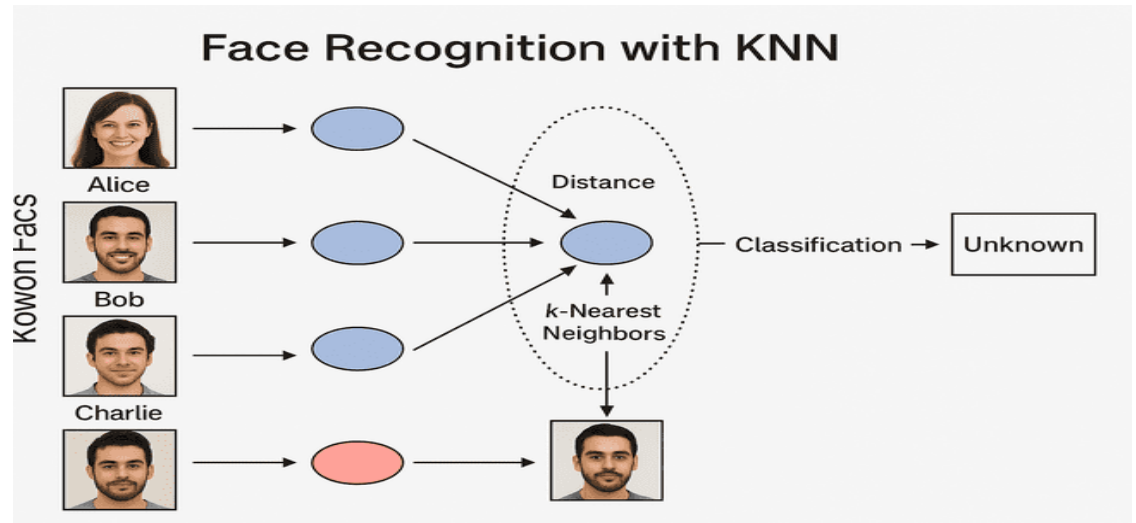


## Use of html and css

### 1.4 KNN algo

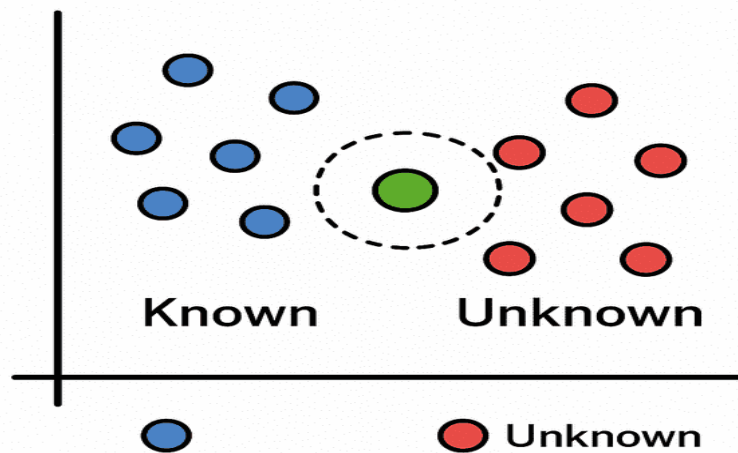
K-Nearest Neighbors (KNN) plays a crucial role in face recognition by classifying detected faces based on their similarity to known encodings. Here's how it works in your project:

- **Face Encoding Storage**
  1. The system maintains a dataset of known faces, each represented as a 128-dimensional vector.
  2. These encodings are stored alongside the corresponding names.
- **Detecting and Encoding a Face**
  1. When a new face appears in the frame, it's detected and encoded using the same method (face\_recognition library).
  2. The encoding is then compared to the stored dataset.
- **Applying KNN for Classification**
  1. The algorithm calculates the distance between the newly detected face and all known faces using Euclidean distance.
  2. It selects the K closest matches (neighbors) based on the smallest distances.
  3. If a majority of neighbors belong to a known person, the face is classified accordingly.
  4. If the distance exceeds a similarity threshold, the face is labeled as "Unknown."
- **Decision & Output**
  1. The system assigns the detected face to the person with the most similar stored encodings.
  2. The name is displayed if the person is recognized; otherwise, it's labeled as "Unknown."



### KNN algorithm

### k-NN for Face Recognition



### Euclidean Distance



### ✓ Summary: System Flow

- **face\_data\_collection.ipynb** – Collect face images of users and build a dataset.
- **face\_recognition.ipynb** – Train a model using PCA + SVM and perform real-time face recognition.
- **haarcascade\_frontalface\_alt.xml** – Used in both steps to detect faces in real-time using OpenCV.

## CHAPTER 2

### Literature Survey/Review of Literature

Face recognition has been a prominent area of research within the fields of computer vision and machine learning for several decades. It has evolved from basic image processing methods to sophisticated deep learning models capable of recognizing faces with high accuracy in complex, real-world environments. This section reviews some of the key contributions and technologies that form the foundation for modern face recognition systems like the one implemented in this project.

#### 2.1 Traditional Approaches

Early face recognition systems primarily relied on feature-based or template-based methods. Some notable approaches include:

##### 2.1.1 Eigenfaces (Turk and Pentland, 1991)

###### Overview:

The Eigenfaces method, proposed by Matthew Turk and Alex Pentland in 1991, was one of the earliest and most influential approaches to automatic face recognition. It introduced the concept of representing face images using Principal Component Analysis (PCA) to reduce the dimensionality of facial data while retaining the most significant features.

###### How It Works:

- Each face image is treated as a high-dimensional vector (e.g., a 100×100 grayscale image becomes a 10,000-dimensional vector).
- PCA is applied to compute the eigenvectors (called *eigenfaces*) of the covariance matrix of the training face dataset.
- These eigenfaces capture the major patterns or directions of variation in the face images.
- A new face image is projected into this *face space* (a lower-dimensional space) by computing its dot product with the top eigenfaces.
- Face recognition is performed by comparing the Euclidean distance or cosine similarity between the projected vectors of the input image and stored images.

###### Advantages:

- Efficient in terms of storage and computation.

- Good for applications with controlled environments (consistent lighting, frontal poses).

#### **Limitations:**

- Very sensitive to lighting, facial expressions, and pose variations.
  - PCA optimizes for variance, not class separation — which makes it less effective in distinguishing between similar faces.
- 

### **2.1.1 Fisherfaces (Belhumeur, Hespanha, and Kriegman, 1997)**

#### **Overview:**

Fisherfaces was introduced to overcome the major weaknesses of Eigenfaces. It uses Linear Discriminant Analysis (LDA) instead of PCA to improve class separability. This method is better at handling illumination changes, facial expressions, and intra-class variability.

#### **How It Works:**

- Initially, PCA is used as a pre-processing step to reduce dimensionality (especially important when the number of training samples is less than the number of pixels).
- Then, LDA is applied to maximize the ratio of between-class scatter to within-class scatter, ensuring better separation of different individuals' faces.
- The result is a set of discriminant vectors (called *fisherfaces*) onto which face images are projected.
- Recognition is performed by comparing the projections of input faces with those of known faces using distance metrics.

#### **Advantages:**

- More robust against lighting variations and intra-class changes (e.g., different facial expressions).
- Specifically optimizes for discriminability between classes (i.e., between different individuals).

#### **Limitations:**

- Assumes linear separability of classes — may struggle with non-linear variations in data.



- Requires that the number of training samples per class be sufficient.

---

### Comparison Summary:

Feature	Eigenfaces (PCA)	Fisherfaces (PCA + LDA)
Focus	Maximize variance	Maximize class separability
Sensitive to Lighting	Yes	Less sensitive
Pose & Expression	Sensitive	More robust
Training Requirements	Fewer samples needed	Needs more samples/class
Computational Speed	Faster	Slightly slower (LDA step)

While these methods were computationally efficient, their performance declined in uncontrolled environments with occlusion, varying lighting, and facial expressions.

## 2.2 Machine Learning-Based Methods

As the limitations of earlier statistical methods like Eigenfaces and Fisherfaces became apparent — particularly under real-world conditions involving varying lighting, expressions, and occlusions — the 2000s witnessed a major shift in face recognition research. This era saw the adoption of machine learning classifiers and handcrafted feature extraction techniques, which provided improved performance by better capturing discriminative facial features.

---

### Popular Machine Learning Classifiers Used:

#### 2.2.1 Support Vector Machines (SVMs):

- **Overview:** SVMs are supervised learning models that find the optimal hyperplane to separate classes with maximum margin.
- **Application in Face Recognition:**
  1. After extracting features from face images, SVMs are used to classify faces by finding boundaries between different individuals.

2. Particularly effective for binary classification tasks (e.g., "match" vs "no match") and extended to multi-class problems using techniques like one-vs-rest.
- **Strengths:** High generalization performance; works well even in high-dimensional spaces.

### 2.2.2 k-Nearest Neighbors (k-NN):

- **Overview:** A simple, instance-based learning algorithm that classifies input based on the majority class among its 'k' closest neighbors in the feature space.
- **Application in Face Recognition:**
  1. Widely used due to its simplicity and non-parametric nature.
  2. Suitable when the dataset is not too large; computationally intensive during prediction.
- **Strengths:** No training time; easy to implement; works well with small datasets.

### 2.2.3 Hidden Markov Models (HMMs):

- **Overview:** Statistical models that represent systems with temporal or sequential dependencies using hidden states.
- **Application in Face Recognition:**
  1. Used for modeling sequences of facial features, such as changes in expression or speech (in video-based face recognition).
  2. Suitable for dynamic face recognition, where the temporal relationship between frames matters.
- **Strengths:** Effective in capturing temporal patterns and transitions.

---

## □ Handcrafted Feature Extraction Techniques:

Before deep learning automated feature learning, systems relied on manually engineered features to describe the key visual characteristics of a face. These descriptors were then fed into the classifiers above.

### 2.2.1 Local Binary Patterns (LBP):

- **How it works:** Encodes the local texture around each pixel by thresholding its neighborhood.
- **Advantages:**
  1. Robust to lighting changes.
  2. Fast and computationally simple.
- **Usage:** Popular for real-time face detection and recognition tasks, especially in low-resolution images.

### 2.2.2 Histogram of Oriented Gradients (HOG):

- **How it works:** Computes the distribution (histogram) of gradient orientations in localized regions of an image.
- **Advantages:**
  1. Captures edge and shape information effectively.
  2. Invariant to small geometric transformations.
- **Usage:** Often used in combination with SVMs for face detection and recognition.

---

### 📌 Limitations of This Generation:

Despite outperforming classical methods, these machine learning techniques were still constrained by their dependence on manually designed features. Their performance often suffered due to:

- Sensitivity to noise, occlusion, and complex backgrounds.
- Lack of ability to automatically learn high-level abstractions from raw data.
- Difficulty in generalizing across different demographics or real-world scenarios.

---

### 🔄 Transition to Deep Learning:

By the early 2010s, deep learning — especially Convolutional Neural Networks (CNNs) — began to replace these traditional pipelines by enabling end-to-end learning directly from pixel values. Deep networks could automatically learn hierarchical feature representations, leading to breakthroughs in accuracy and scalability.

---

✓ **Summary:**

Aspect	Classical (PCA, LDA)	ML-based (SVM, k-NN, LBP/HOG)	Deep Learning
Feature Extraction	Manual	Manual	Automatic
Classifier	Statistical	Machine Learning	Deep Neural Networks
Performance	Limited	Improved	State-of-the-art
Robustness	Low	Moderate	High

## 2.3 Deep Learning and CNNs

With the rise of deep learning in the 2010s, face recognition technology saw a monumental leap in accuracy and robustness. The advent of Convolutional Neural Networks (CNNs) transformed the field, as CNNs were capable of automatically learning hierarchical features from raw image data, eliminating the need for manual feature extraction methods. Below is an in-depth exploration of the key developments in this period:

---

### 2.3.1. DeepFace (Taigman et al., 2014)

📖 **Overview:**

DeepFace, developed by Facebook researchers in 2014, is one of the pioneering deep learning models for face recognition. It marked a significant shift in the face recognition paradigm, leveraging CNNs for the first time to achieve near human-level accuracy.

🔧 **How It Works:**

- **Architecture:** DeepFace used a deep CNN model based on a 9-layer network to learn facial representations directly from raw pixel data. It employed alignment techniques to normalize faces before processing, which helped mitigate issues related to pose and lighting variations.

- **Training:** DeepFace was trained on a large-scale dataset of labeled faces (4 million images of 4,000 different people), enabling it to generalize well across diverse subjects.
- **Accuracy:** On the LFW (Labeled Faces in the Wild) dataset, DeepFace achieved an accuracy of 97.25%, which was considered human-level performance and a major milestone in the field.

#### ✓ **Significance:**

- DeepFace demonstrated that deep CNNs could automatically learn robust features for face recognition, eliminating the reliance on hand-crafted features like LBP or HOG.
  - It set the stage for further research in face recognition using CNNs, making deep learning the new standard in the field.
- 

### 2.3.2. FaceNet (Schroff et al., 2015)

#### **Overview:**

FaceNet, introduced by Google researchers in 2015, was a groundbreaking model that redefined the way face recognition was approached by introducing embedding learning. It utilized CNNs to map faces to a compact Euclidean space, where face similarity could be easily measured.

#### **How It Works:**

- **Unified Embedding:** FaceNet created a fixed-length vector embedding for each face image. These embeddings were structured such that the Euclidean distance between two embeddings directly correlated with the degree of similarity between the corresponding faces.
- **Triplet Loss:** One of FaceNet's key innovations was the use of triplet loss. A triplet consists of an anchor image, a positive image (same identity), and a negative image (different identity). The goal of triplet loss was to ensure that the anchor image's embedding was closer to the positive image's embedding than to the negative image's embedding.
- **Training:** FaceNet was trained on a vast dataset of over 200 million face images, allowing it to recognize faces from diverse sources, including multiple angles, lighting conditions, and occlusions.

### ✓ Significance:

- The embedding approach introduced by FaceNet enabled more efficient face comparison, as face recognition could now be reduced to a simple distance calculation between embeddings, improving speed and scalability.
  - The use of triplet loss further enhanced the model's ability to learn discriminative features for face recognition.
- 

### 2.3.3. Dlib and face\_recognition Library

#### 📖 Overview:

Dlib and its associated face\_recognition library have become essential tools in the face recognition community due to their simplicity and accessibility. Dlib provides pre-trained models built on ResNet-based architectures, and the face\_recognition library offers an easy-to-use interface for face detection and recognition.

#### 🔧 How It Works:

- **Dlib's Face Recognition Model:** Dlib's face recognition model, built on a ResNet architecture, is a CNN-based approach that performs face detection, landmark localization, and face recognition with high accuracy.
  1. The model is trained on a large dataset of faces, and it provides outputs like face embeddings, which can then be compared to identify individuals.
- **face\_recognition Library:** The face\_recognition library simplifies the process of using Dlib's face recognition tools, allowing users to easily load images, detect faces, and perform recognition tasks. It also supports face landmarks, which can be used for alignment, and offers pre-trained models for efficient inference.

### ✓ Significance:

- **Accessibility:** Dlib and face\_recognition made powerful face recognition models more accessible to developers and researchers by providing easy-to-use tools with pre-trained models.
- **Versatility:** These libraries offer high-quality, fast face recognition and can be easily integrated into real-time applications such as security systems, user authentication, and even social media platforms.

- **Community Support:** The open-source nature of Dlib and the face\_recognition library has fostered a strong community of developers contributing to their enhancement.

---

### **Impact on Face Recognition:**

The developments outlined above significantly boosted the accuracy, efficiency, and robustness of face recognition systems. Some key improvements include:

- **Higher accuracy** under difficult conditions (e.g., pose changes, varying lighting, occlusions).
- **Scalability:** These models could recognize faces from millions of identities while maintaining high performance.
- **Real-time applications:** With models like Dlib and face\_recognition, real-time face recognition became feasible, even on less powerful hardware (e.g., smartphones, embedded systems).

### **Benefits of Deep Learning-Based Face Recognition:**

- **Robustness:** CNNs can handle large variations in facial expressions, occlusions, and lighting conditions.
- **End-to-End Learning:** Unlike earlier methods that required manual feature extraction, deep learning models can learn directly from raw image pixels.
- **Real-World Applicability:** These models can be deployed in complex, uncontrolled environments like social media platforms, surveillance systems, and user authentication.

---

### **Summary of Key Models:**

Model	Year	Key Innovation	Achievements
DeepFace	2014	Deep CNN model for face recognition	Achieved human-level accuracy on LFW
FaceNet	2015	Unified embedding space and triplet loss for face recognition	Revolutionized face similarity and recognition

Model	Year	Key Innovation	Achievements
Dlib	2014	ResNet-based face detection and recognition model	Efficient, real-time face recognition tools
face_recognition	2015	Simplified Dlib interface for easy face recognition	Made state-of-the-art face recognition accessible

## 2.4 Real-Time Face Recognition

Real-time face recognition systems have become more accessible and efficient thanks to a combination of powerful tools and libraries. By integrating OpenCV, dlib/face\_recognition, and NumPy/Pandas, developers have been able to create lightweight face recognition systems that perform effectively on consumer-grade hardware. Here's an in-depth look at these tools and how they work together in real-time systems:

---

### 2.4.1. OpenCV (Open Source Computer Vision Library)

#### Overview:

OpenCV is one of the most widely used open-source libraries for computer vision tasks. It offers a vast collection of functions and tools for image processing, computer vision, and machine learning.

#### Role in Real-Time Face Recognition:

- **Video Capture:** OpenCV provides functions to easily access the camera feed for real-time video capture, enabling systems to continuously stream images or frames.
  1. `cv2.VideoCapture()` is commonly used to capture video input from a webcam or other video sources.
- **Real-Time Image Processing:** OpenCV's image processing functions (such as resizing, color space conversion, and Gaussian blurring) help improve the quality of images and prepare them for face detection.
  1. **Face Detection:** OpenCV includes pre-trained Haar cascades and HOG-based detectors, which can be used to detect faces in each frame.

#### Advantages:



- **Open-Source and Fast:** OpenCV is highly optimized for performance and can run efficiently on consumer-grade hardware (even mobile devices).
  - **Flexibility:** It supports multiple backends for accessing cameras and processing video data, allowing for easy integration into various applications.
- 

## 📁 2.4.2. dlib / face\_recognition

### 📖 Overview:

- dlib is a toolkit containing machine learning algorithms and tools for solving real-world problems in areas such as computer vision and image processing.
- The face\_recognition library, built on top of dlib, provides a simple interface to work with dlib's powerful face recognition models.

### 🔧 Role in Real-Time Face Recognition:

- **Face Detection:**
  - dlib provides HOG-based face detectors and CNN-based detectors for more accurate results. These detectors are capable of identifying faces in images or video frames.
- **Face Encoding (Feature Extraction):**
  - Once a face is detected, dlib's facial landmark detector can locate key facial features (eyes, nose, mouth, etc.).
  - The face\_recognition library uses dlib's deep learning model to generate 128-dimensional embeddings for each face, which represent the unique features of the face.
  - These embeddings can be compared to recognize or verify individuals.
- **Real-Time Matching:**
  - Using embeddings generated by dlib, face\_recognition can match the detected face to known faces stored in a database by comparing Euclidean distances between embeddings.

### ✅ Advantages:

- **High Accuracy:** dlib and face\_recognition offer state-of-the-art accuracy, even in challenging conditions like varied poses and lighting.
  - **Speed:** These libraries are optimized for real-time processing and can run effectively on CPUs without requiring high-end GPUs.
  - **Simplicity:** The face\_recognition library provides a high-level API for face detection, encoding, and recognition, making it easy for developers to integrate face recognition into their applications.
- 

### 2.4.3. NumPy / Pandas

#### Overview:

- **NumPy** is a powerful library for numerical computing in Python, providing support for large multidimensional arrays and matrices.
- **Pandas** is a data manipulation library built on top of NumPy, providing powerful data structures like DataFrames for handling and analyzing large datasets.

#### Role in Real-Time Face Recognition:

- **Data Handling and Performance Optimization:**
  1. **NumPy** allows for efficient handling of large arrays and matrices, which is essential when working with high-dimensional embeddings (like the 128-dimensional face embeddings from dlib).
  2. **Pandas** can be used to manage and manipulate structured datasets (e.g., storing known faces and their associated labels) for face recognition systems.
  3. **Vector Operations:** NumPy provides efficient operations on vectors (like computing Euclidean distances between face embeddings), which is crucial for fast face matching and recognition.
  4. **Real-Time Data Tracking:** Pandas can track real-time data, like logging detected faces or tracking recognized individuals during a session.

#### Advantages:

- **Speed:** NumPy's vectorized operations allow for quick calculations, ensuring that face matching and recognition happen in real-time.

- **Ease of Use:** Pandas allows for easy data manipulation and storage, making it convenient for handling databases of known faces.
- 

### **Combining the Tools for Real-Time Face Recognition**

By combining OpenCV, dlib/face\_recognition, and NumPy/Pandas, developers can create real-time face recognition systems that are both efficient and scalable. Here's a breakdown of how the tools work together:

- **Capture video frames** from the camera using OpenCV.
- **Detect faces** in each frame using dlib's face detectors.
- **Extract features (face embeddings)** from detected faces using dlib's face\_recognition library.
- **Compare embeddings** (using NumPy/Pandas) to a database of known faces for recognition.
- **Display results** (e.g., showing the name of the recognized person on the screen) in real-time.

### **Benefits of Combining These Tools:**

- **Lightweight and Efficient:** The integration of these tools allows developers to create fast, low-resource applications suitable for real-time deployment on consumer-grade hardware (e.g., laptops, smartphones).
  - **Accuracy and Robustness:** Using state-of-the-art libraries like dlib and face\_recognition ensures that the system remains accurate even under challenging conditions like pose variations, occlusion, and lighting changes.
  - **Open Source:** These libraries are open-source, making them accessible and cost-effective for developers and businesses.
- 

### **Applications of Real-Time Face Recognition Systems:**

- **Security Systems:** Real-time surveillance systems for identifying intruders or verifying authorized personnel.
- **Access Control:** Systems for verifying individuals at doors or gates.

- **Customer Interaction:** Personalized experiences in retail stores or kiosks that use facial recognition for identification or payment verification.
- **Mobile Authentication:** Face unlock features on smartphones and tablets.

## CHAPTER 3

### Software Requirement Specification

The Software Requirement Specification (SRS) for a Face Recognition System outlines the comprehensive functionalities, technical specifications, and operational conditions needed for the system's development. It serves as a guide for the entire project, ensuring all aspects of the face recognition process are covered and understood. Below is a detailed version of the SRS document:

---

#### 3.1. Introduction

##### **Purpose:**

The purpose of this project is to design and implement a real-time face recognition system using Python. This system will be able to identify individuals based on stored facial images captured through a webcam. The system will be able to detect faces, encode them, and compare them with a pre-existing dataset to match and identify the individual.

##### **Scope:**

This face recognition system is designed primarily for educational and prototyping purposes, with the potential to be extended for practical applications such as:

- Attendance systems in educational institutions.
- Security verification in access control systems.
- User authentication in mobile and desktop applications.

The system will be capable of:

- Detecting faces from a live video feed.
  - Matching detected faces with a pre-encoded dataset of known individuals.
  - Displaying the names of identified individuals on the screen in real time.
  - Labeling faces as “Unknown” when no match is found.
- 

#### 3.2. Functional Requirements

##### 3.2.1. Face Registration:

- **Description:** The system should allow users to load known individuals' images into the system. These images will be encoded and stored for future matching.
- **Input:** A set of images from known individuals (JPEG, PNG formats).
- **Output:** Encoded facial data stored in a dataset.

#### 3.2.2. Face Detection:

- **Description:** The system must detect visible faces in a live video stream captured by the webcam.
- **Input:** Video stream from the webcam.
- **Output:** Bounding box coordinates for detected faces.

#### 3.2.3. Face Encoding:

- **Description:** The system should convert the detected faces into a 128-dimensional encoding vector that represents the unique features of the face.
- **Input:** Detected faces from the video feed.
- **Output:** 128-dimensional face encoding.

#### 3.2.4. Face Matching:

- **Description:** The system must compare the face encodings of detected faces with the stored encodings of known individuals.
- **Input:** The 128-dimensional encoding of the detected face.
- **Output:** Recognition result indicating whether the detected face matches any of the stored faces, along with the individual's name.

#### 3.2.5. Labeling:

- **Description:** Upon recognition, the system should display the name of the recognized individual above the corresponding face in real-time.
- **Input:** Recognition result (matching individual's name).
- **Output:** Displayed name on the video stream above the detected face.

#### 3.2.6. Unknown Detection:

- **Description:** If the system does not recognize the detected face, it should label the face as "Unknown."
  - **Input:** Faces that do not match any stored face encoding.
  - **Output:** "Unknown" label displayed on the video feed.
- 

### 3.3. Non-Functional Requirements

#### 3.3.1. Performance:

- **Description:** The system should be able to process at least 15–20 frames per second to ensure smooth and real-time face recognition.
- **Expectation:** Efficient processing on an average consumer-grade computer.

#### 3.3.2. Scalability:

- **Description:** The system should allow easy addition of new individuals without requiring a complete retraining of the model or system.
- **Expectation:** Adding new faces should involve just loading new images into the system and encoding them.

#### 3.3.3. Reliability:

- **Description:** The face recognition accuracy should remain high despite variations in lighting, angles, or partial occlusion of faces.
- **Expectation:** System should perform consistently under different real-world conditions.

#### 3.3.4. Usability:

- **Description:** The system should have an intuitive and simple interface, ideally through a console application or basic graphical user interface (GUI).
- **Expectation:** Users should be able to interact easily with the system, register faces, and monitor recognition in real-time.

#### 3.3.5. Portability:

- **Description:** The software should be portable and run on any system with Python 3.x installed (Windows, Linux, or Mac OS).

- **Expectation:** The system should be cross-platform without any dependency on specific hardware configurations.
- 

### 3.4. Software Requirements

#### **Programming Language:**

- Python 3.7 or later will be used for implementing the system, as it is compatible with the necessary libraries for face recognition.

#### **Libraries/Modules:**

- `face_recognition`: Provides easy-to-use functions for face detection and encoding.
- `OpenCV (cv2)`: Handles video capture and image processing for real-time video streams.
- `NumPy`: Used for efficient numerical operations, particularly during face matching using encoded feature vectors.
- `os`: Facilitates file and directory operations such as loading images from the system.

#### **IDE/Editor:**

- Any Python-compatible Integrated Development Environment (IDE) or editor, such as:
    1. Visual Studio Code
    2. Jupyter Notebook
    3. PyCharm
- 

### 3.5. Hardware Requirements

- **Processor:** Intel i5 or higher (recommended).
- **RAM:** Minimum of 4 GB (8 GB recommended).
- **Camera:** A built-in or external webcam capable of providing real-time video feed (minimum 720p resolution recommended).



- Storage: Minimum of 500 MB to store dependencies and datasets of known individuals' images.
- 

### **3.6. External Interface Requirements**

User Interface:

- Terminal/Console output is expected by default.
  1. The system will display the video feed with labeled names above recognized faces.
  2. Optionally, a basic GUI could be developed for a more user-friendly interface.

**Hardware Interface:**

- Webcam Interface: The system will interact with the webcam to capture video input using OpenCV.

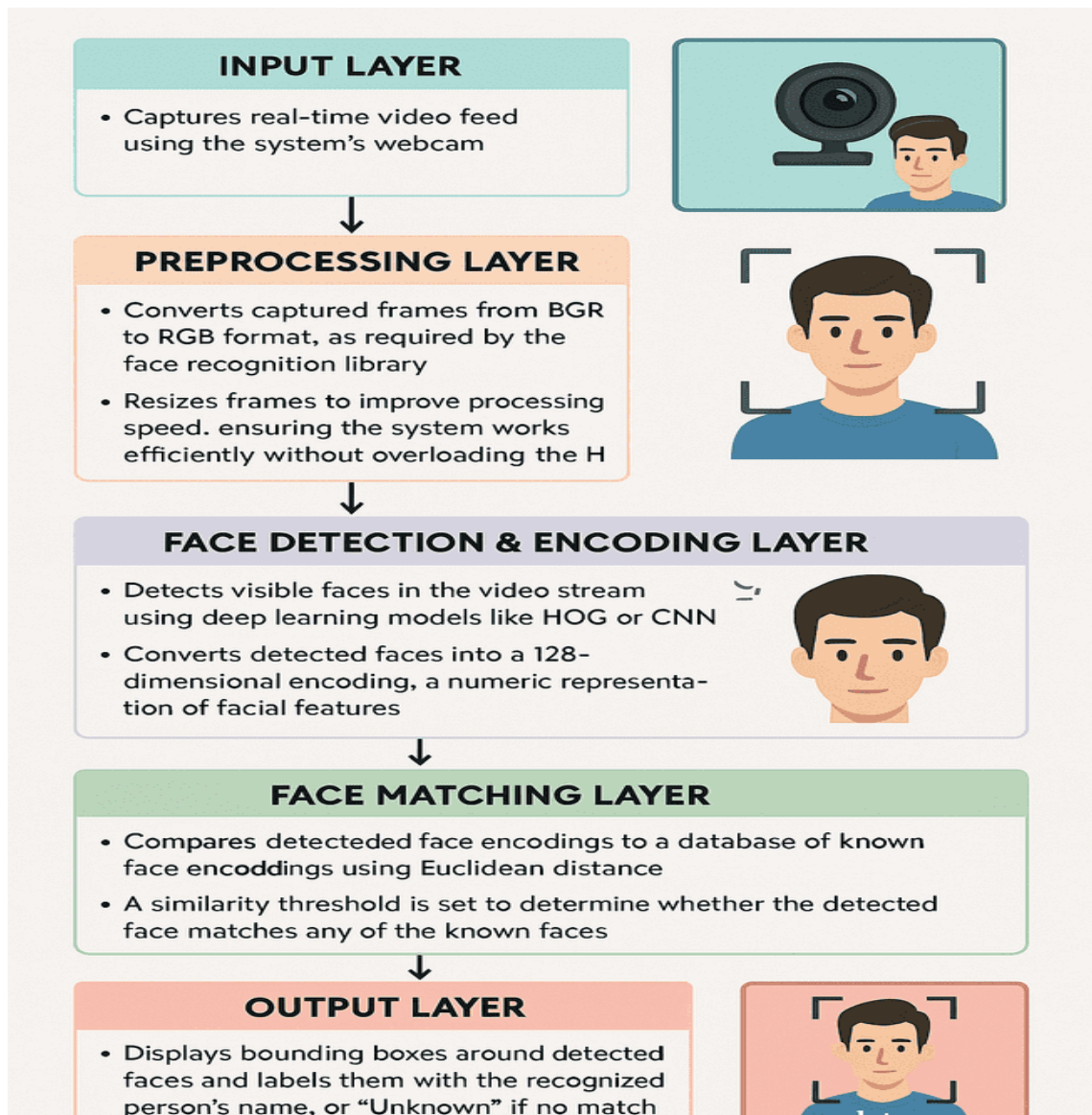
**Software Interface:**

- The system will interact with the operating system via the Python libraries like os for file management and OpenCV for camera interaction.

## CHAPTER 4

### SOFTWARE DESIGN

The **Software Design** phase for the Face Recognition system is a crucial step in creating a robust, maintainable, and extensible application. The design of this system ensures that each module and component performs a specific function to allow for real-time face recognition with minimal latency



### System architecture

#### 4.1. System Architecture

The system follows a **modular** and **layered architecture**, where each layer focuses on a specific task:

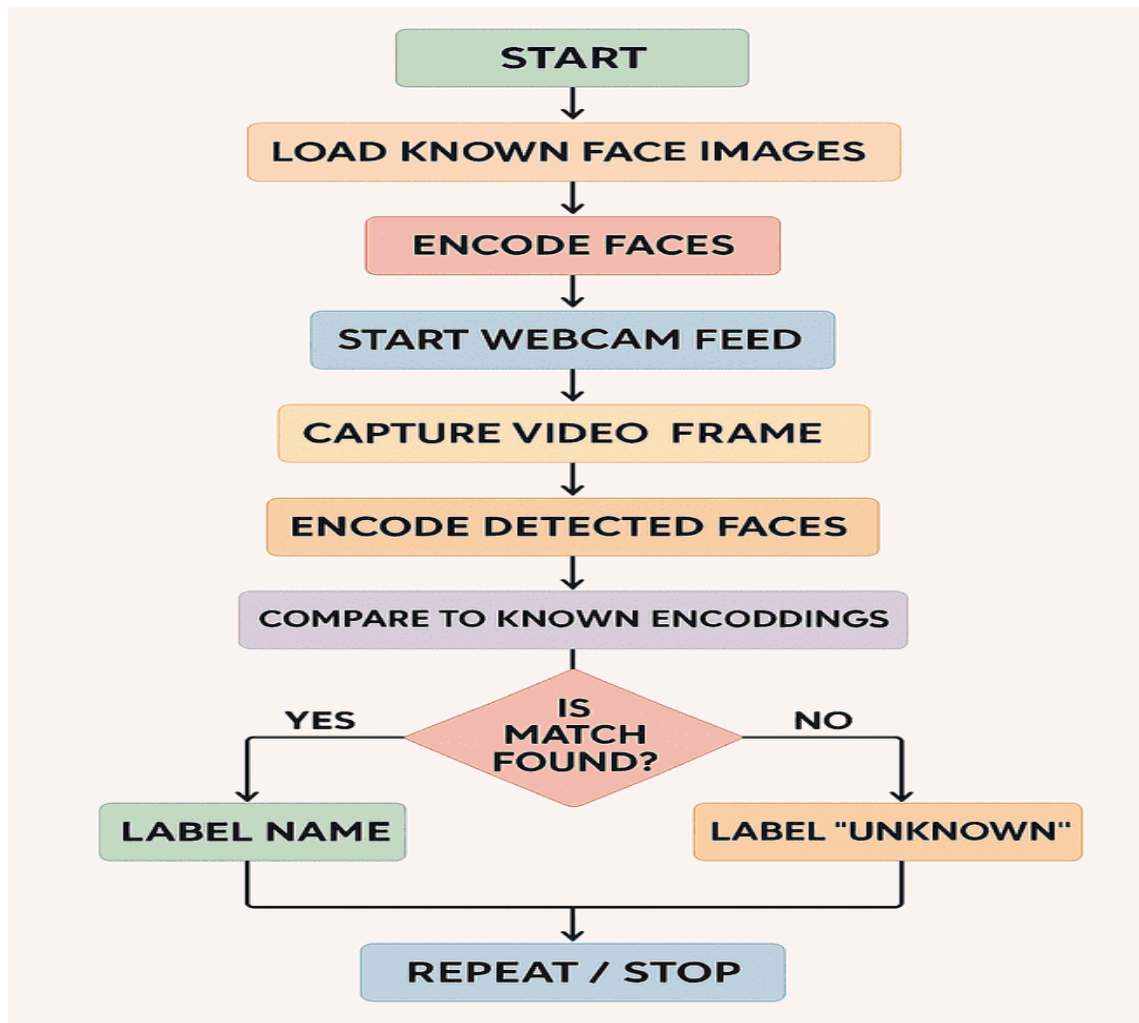
- **Input Layer:**
  1. Captures real-time video feed using the system's webcam.
- **Preprocessing Layer:**
  1. Converts the captured frames from BGR to RGB format, as required by the face recognition library.
  2. Resizes frames to improve processing speed, ensuring that the system works efficiently without overloading the hardware.
- **Face Detection & Encoding Layer:**
  1. Detects visible faces in the video stream using deep learning models like HOG (Histogram of Oriented Gradients) or CNN (Convolutional Neural Networks).
  2. Converts the detected faces into a 128-dimensional encoding, which is a numeric representation of the facial features.
- **Face Matching Layer:**
  1. Compares the detected face encodings to a database of known face encodings using Euclidean distance.
  2. A similarity threshold is set to determine whether the detected face matches any of the known faces.
- **Output Layer:**
  1. Displays bounding boxes around detected faces and labels them with the recognized person's name, or "Unknown" if no match is found.
  2. The final annotated frame is displayed to the user in real-time.

#### 4.2. Module Breakdown

The software is divided into the following core modules:

Module	Description
load_known_faces()	Loads images of known individuals and computes their encodings.
capture_video()	Initializes the webcam and continuously captures video frames.
detect_faces()	Uses the face_recognition library to locate faces in each frame.
match_faces()	Compares the newly captured face encodings with the known encodings.
display_results()	Draws bounding boxes and labels using OpenCV and displays the video stream.

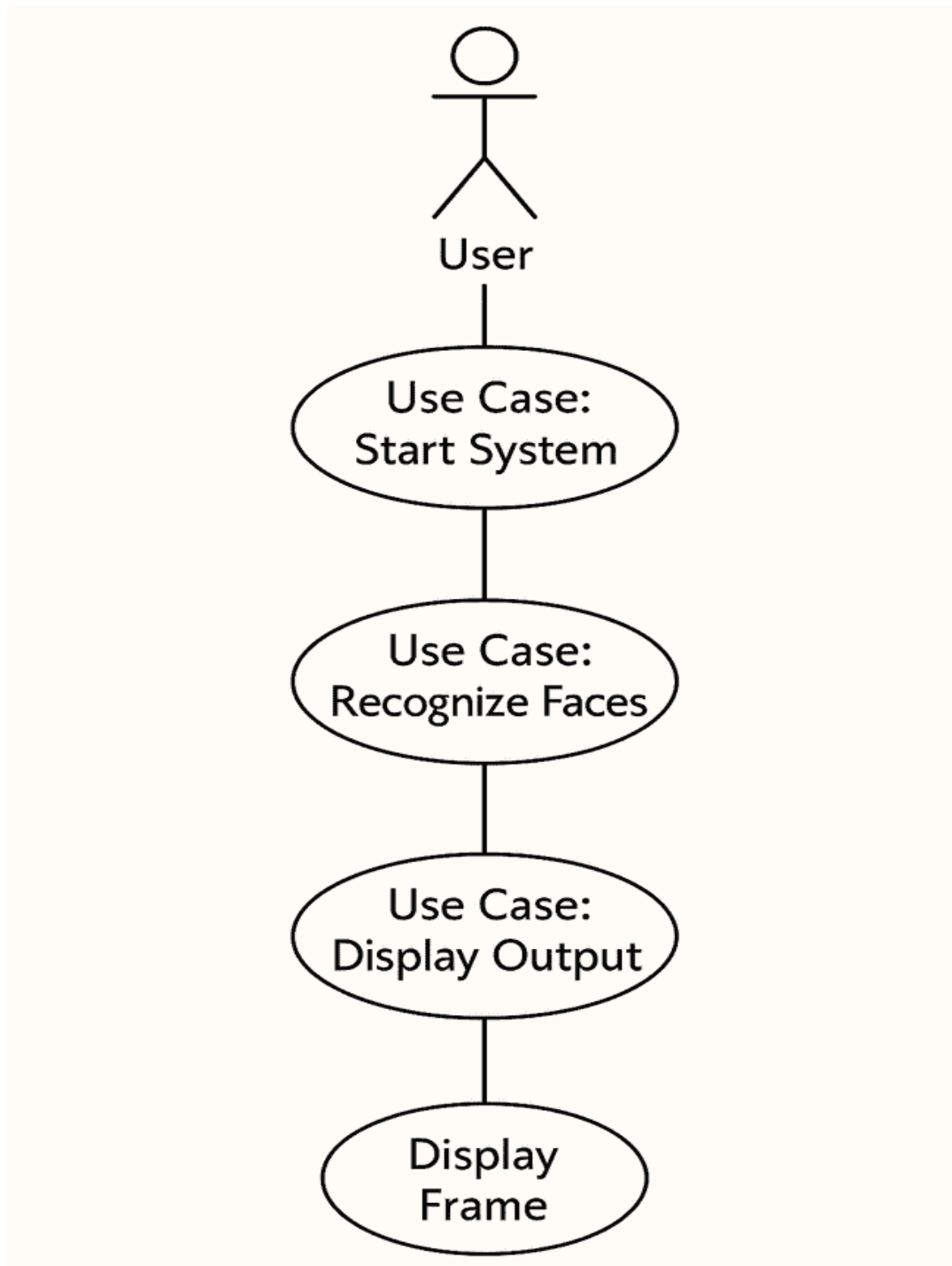
#### 4.3. Data Flow Diagram (DFD - Level 1)



#### DFD

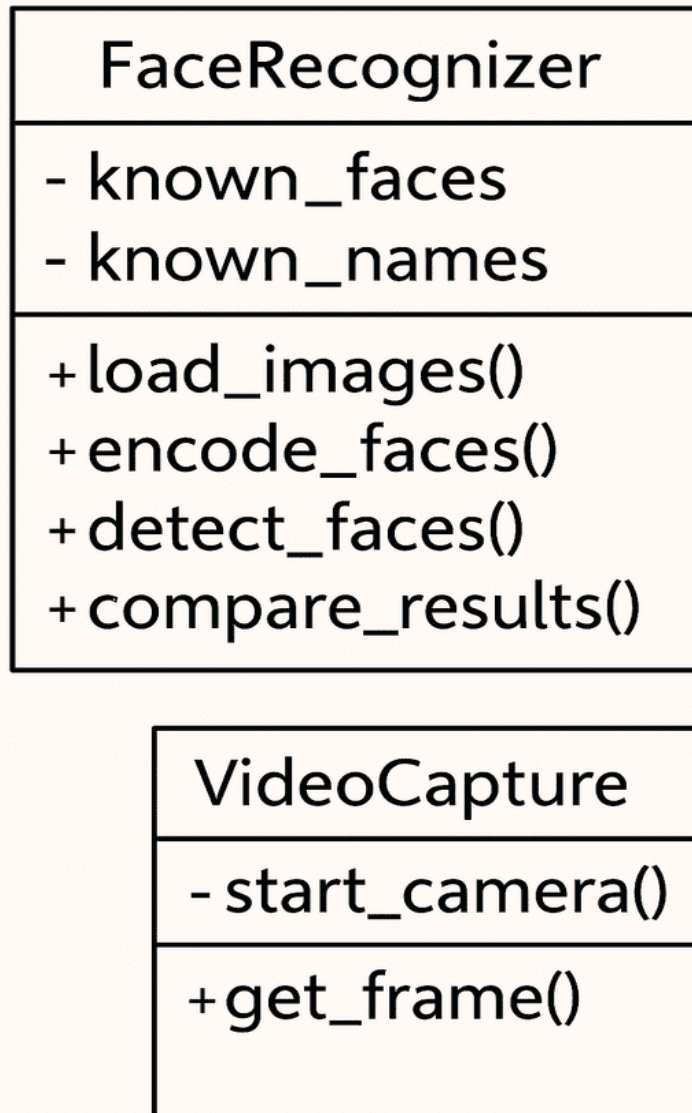
#### 4.4. UML Diagrams

##### a. Use Case Diagram



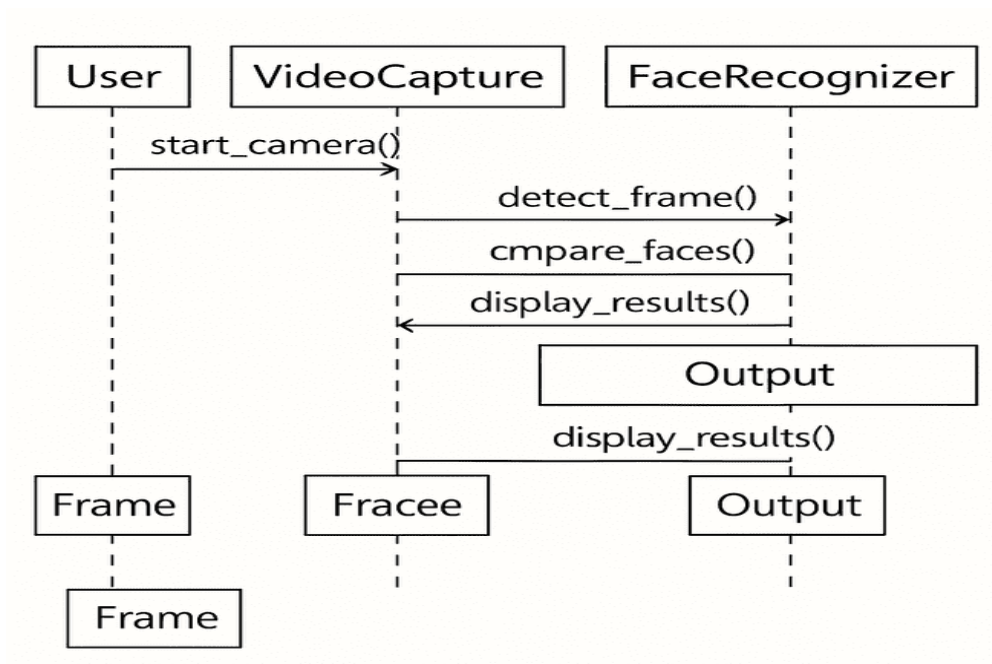
Use Case Diagram

##### b. Class Diagram



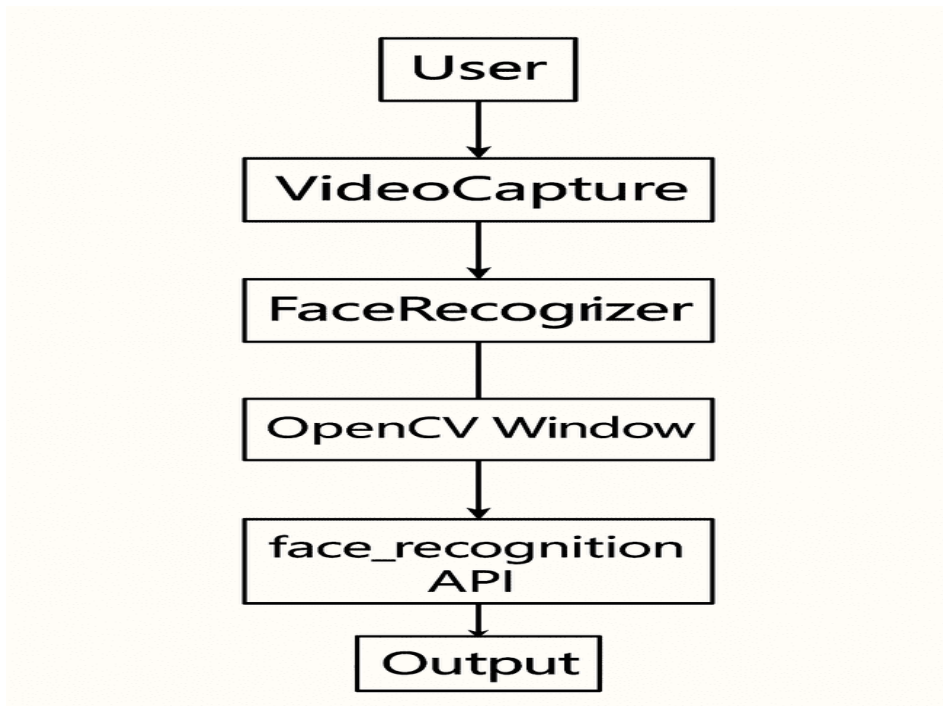
### Class Diagram

#### c. Sequence Diagram



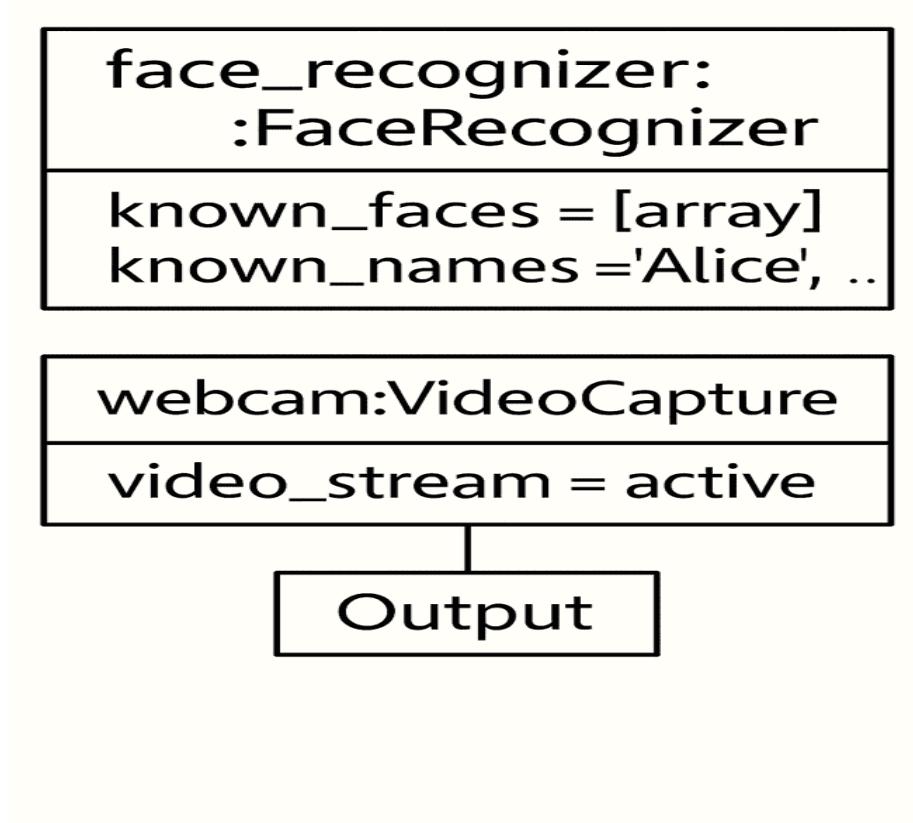
Sequence Diagram

**d. Collaboration Diagram**



Collaboration Diagram

#### e. Object Diagram



### Object Diagram

#### 4.5. Algorithm Overview

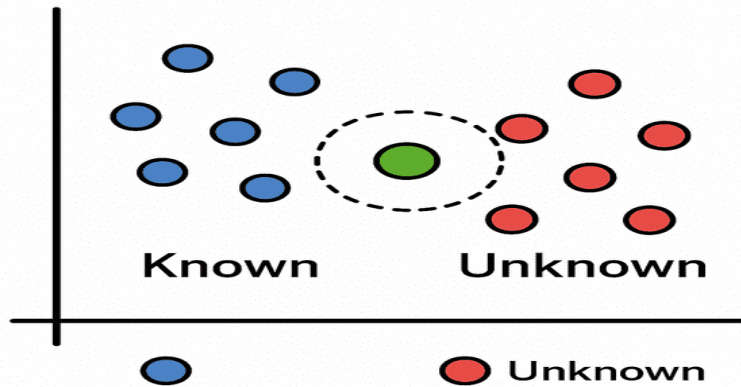
The core **face recognition algorithm** follows these steps:

- **Load known images:** Images of known individuals are loaded into the system.
- **Encode known faces:** The system generates facial encodings for each known individual.
- **Capture frame:** A frame is captured from the webcam in real-time.
- **Detect faces:** The faces in the captured frame are detected.
- **Generate encodings:** Each detected face is encoded into a numpy array
- **Compare faces:** The system compares the generated face encodings with those of known individuals.

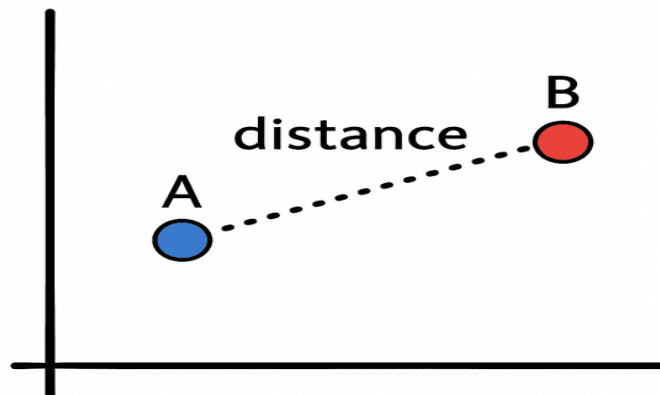


- **Label faces:** If a match is found, the name of the person is displayed; otherwise, the face is labeled as "Unknown."
- **Repeat:** The system continuously repeats this process in real-time.

### k-NN for Face Recognition



### Euclidean Distance



## 6. Design Considerations

- **Real-Time Efficiency:** The system uses optimized algorithms, such as HOG and CNN, to maintain smooth performance even with live video input.
- **Modularity:** The design allows for the easy addition of new features, such as database integration or a graphical user interface (GUI).

- **Extensibility:** New faces can be added to the dataset without the need to retrain the system, ensuring easy scalability.

This design ensures that the Face Recognition system is efficient, easy to extend, and capable of handling real-time input effectively.

## CHAPTER 5

### Software and Hardware requirements

This section outlines the software and hardware needed to develop, run, and test the Face Recognition system effectively.

---

#### 5.1 Software Requirements

Component	Details
Operating System	Windows 10/11, Linux (Ubuntu), or macOS
Programming Language	Python 3.7 or later
IDE/Editor	Visual Studio Code, PyCharm, Jupyter Notebook (optional)
Libraries/Frameworks	-face_recognition -OpenCV(cv2) -NumPy - os
Face Recognition Model	Built-in deep learning model from face_recognition (ResNet-based)
Installation Tools	pip, virtualenv (for managing packages and environments)
Version Control	Git (if used for collaboration or backup)
Other Tools	Command Line or Terminal for running scripts

---

#### 5.2 Hardware Requirements

Component	Minimum Requirement	Recommended Requirement
Processor	Intel i3 or equivalent	Intel i5/i7 or AMD Ryzen 5/7
RAM	4 GB	8 GB or more

Component	Minimum Requirement	Recommended Requirement
<b>Storage</b>	500 MB for software and image data	1 GB+ for additional face datasets/logs
<b>Camera</b>	Built-in or external webcam (720p HD or Full HD webcam for better minimum)	accuracy
<b>GPU (optional)</b>	Not required	NVIDIA GPU (if deep learning acceleration needed)

---

### 5.3 Optional Enhancements

For scalability and future improvements, additional tools or platforms could be considered:

- **Cloud Storage** (e.g., AWS S3) for storing facial data or logs
- **Database Integration** (e.g., SQLite, PostgreSQL) to manage user profiles
- **GUI Frameworks** (e.g., Tkinter, PyQt) to add a user-friendly interface

## CHAPTER 6

### Coding /Code Templates

This section outlines the source code used in the Face Recognition project. It includes the structure of each module, key functions, and an explanation of the classes and methods used. Since this is not a database-driven project, the table schema and stored procedures are not applicable.

---

#### 6.1 Project Structure (Files Overview)

Face-Recognition-/

```
|— images/          # Folder containing images of known individuals
|   |— person1.jpg
|   |— person2.jpg
|— main.py          # Main execution script
|— requirements.txt # Python dependencies
```

---

#### 6.2 main.py – Core Script

Key Libraries Used

```
import face_recognition
import cv2
import numpy as np
import os
```

---

#### 6.3 Code Explanation and Classes

Although the script is function-based rather than object-oriented, for clarity, we can represent the core logic through a conceptual class-based design.

---

Class: FaceRecognizer

Description:

Handles loading of known images, face detection, encoding, and recognition.

---

Methods:

`__init__()`

Functionality: Initializes empty lists for face encodings and names.

`def __init__(self):`

`self.known_face_encodings = []`

`self.known_face_names = []`

---

`load_known_faces(images_folder: str)`

Functionality: Loads images from a folder and encodes them.

Input:

- `images_folder` – Path to the directory containing known face images

Output:

- Updates `self.known_face_encodings` and `self.known_face_names`

`def load_known_faces(self, images_folder):`

`for filename in os.listdir(images_folder):`

`image = face_recognition.load_image_file(f"{images_folder}/{filename}")`

`encoding = face_recognition.face_encodings(image)[0]`

`self.known_face_encodings.append(encoding)`

`self.known_face_names.append(os.path.splitext(filename)[0])`

---

`recognize_faces()`

Functionality: Captures video from webcam, detects faces, and matches them to known encodings.

Input:

- `None` (reads from webcam)

Output:

- Displays video frames with labeled faces in real time

`def recognize_faces(self):`

`video_capture = cv2.VideoCapture(0)`

```

while True:
    ret, frame = video_capture.read()
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
    rgb_small_frame = small_frame[:, :, :-1]

    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame,
    face_locations)

    face_names = []
    for face_encoding in face_encodings:
        matches = face_recognition.compare_faces(self.known_face_encodings,
        face_encoding)
        name = "Unknown"
        face_distances = face_recognition.face_distance(self.known_face_encodings,
        face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = self.known_face_names[best_match_index]
        face_names.append(name)

    for (top, right, bottom, left), name in zip(face_locations, face_names):
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)

```

```
cv2.putText(frame, name, (left + 6, bottom - 6), cv2.FONT_HERSHEY_DUPLEX, 1.0,
(255, 255, 255), 1)
```

```
cv2.imshow('Video', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
video_capture.release()
cv2.destroyAllWindows()
```

---

#### 6.4 Code Template Summary

Function	Purpose	Input	Output
<code>__init__()</code>	Initializes data containers	None	Class variables
<code>load_known_faces()</code>	Reads and encodes known face images	Folder path	Known encodings + names
<code>recognize_faces()</code>	Real-time face detection & recognition	Webcam frames	Live labeled video feed



## CHAPTER 7

### TESTING

#### 7. Testing

Testing ensures that the face recognition system works as expected under various conditions. Both **Black Box Testing** and **White Box Testing** were applied.

##### 7.1 Black Box Testing

###### Definition:

Black box testing focuses on system functionality without knowing the internal code structure. It validates input/output behavior.

###### ✓ Test Case 1: Known Face Recognition

**Test Case ID** TC\_BB\_01

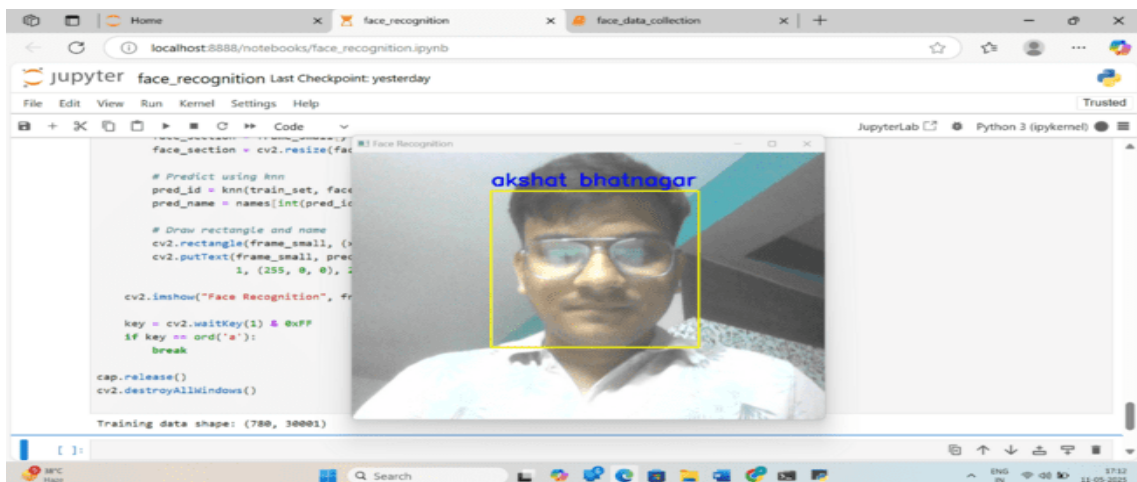
**Description** System should recognize a known face and display the correct label.

**Input** Person with a known face in front of webcam.

**Expected Output** Name of the person (e.g., “Akshat”) displayed on screen.

**Actual Output** ✓ Name displayed correctly.

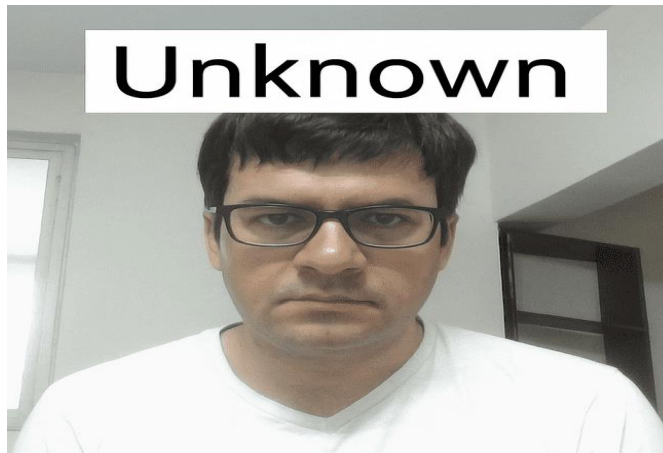
**Status** Pass



#### Known Face Recognition

### ✓ Test Case 2: Unknown Face Handling

<b>Test Case ID</b>	<b>TC_BB_02</b>
<b>Description</b>	System should detect a face not in the dataset and label it as "Unknown".
<b>Input</b>	New person not added to training dataset.
<b>Expected Output</b>	"Unknown" displayed under detected face.
<b>Actual Output</b>	✓ Correctly labeled as Unknown.
<b>Status</b>	Pass



**Unknown Face Handling**

## 7.2 White Box Testing

### Definition:

White box testing verifies internal code structure, logic flow, and decision branches by examining the source code directly.

### ✓ Test Case 3: Encoding Function Logic

<b>Test Case ID</b>	<b>TC_WB_01</b>
<b>Description</b>	Ensure that all images in the images/ directory are encoded properly and added to known_faces list.
<b>Tested Method</b>	load_known_faces()
<b>Expected</b>	For each image, face encoding is generated and name is stored.

**Test Case ID** TC\_WB\_01

**Behavior**

**Test Technique** Debugging and printing list lengths after encoding.

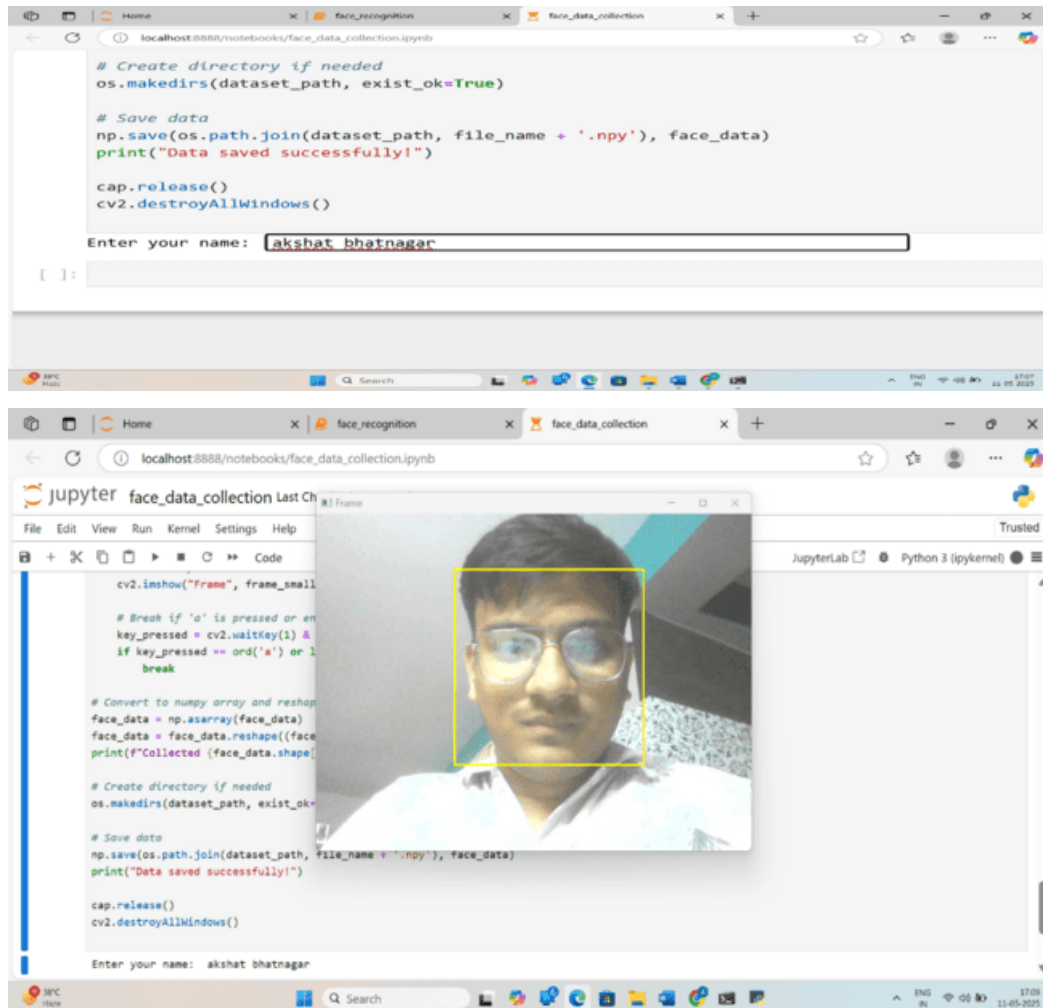
**Actual**

**Behavior**

✓ All valid images encoded and loaded successfully.

**Status**

Pass



### Encoding Function Logic

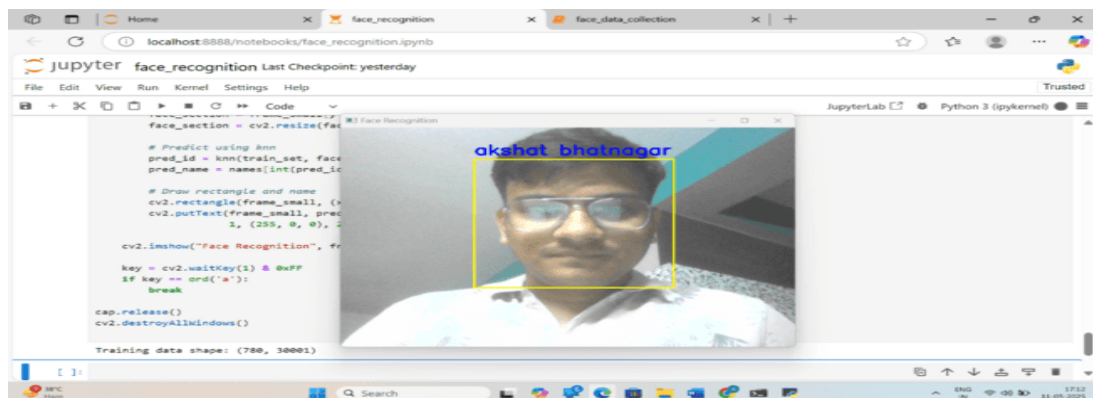
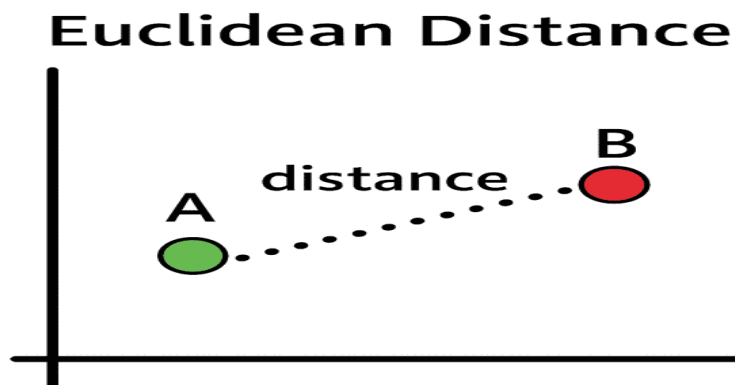
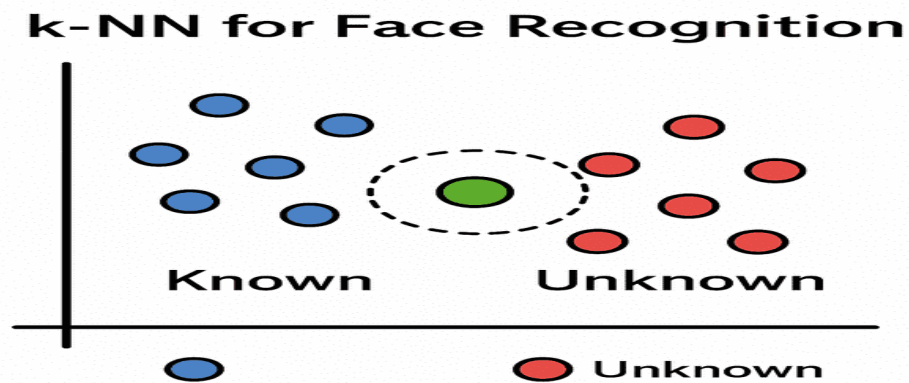
✓ **Test Case 4: Face Match Logic**

**Test Case ID** TC\_WB\_02

**Description**

Test correctness of face distance calculation and match threshold.

Test Case ID	TC_WB_02
Tested Method	compare_faces() + face_distance()
Expected Output	Closest match is selected and label is displayed.
Observed Output	<input checked="" type="checkbox"/> Correct label shown for best match.
Status	Pass



## CHAPTER 8

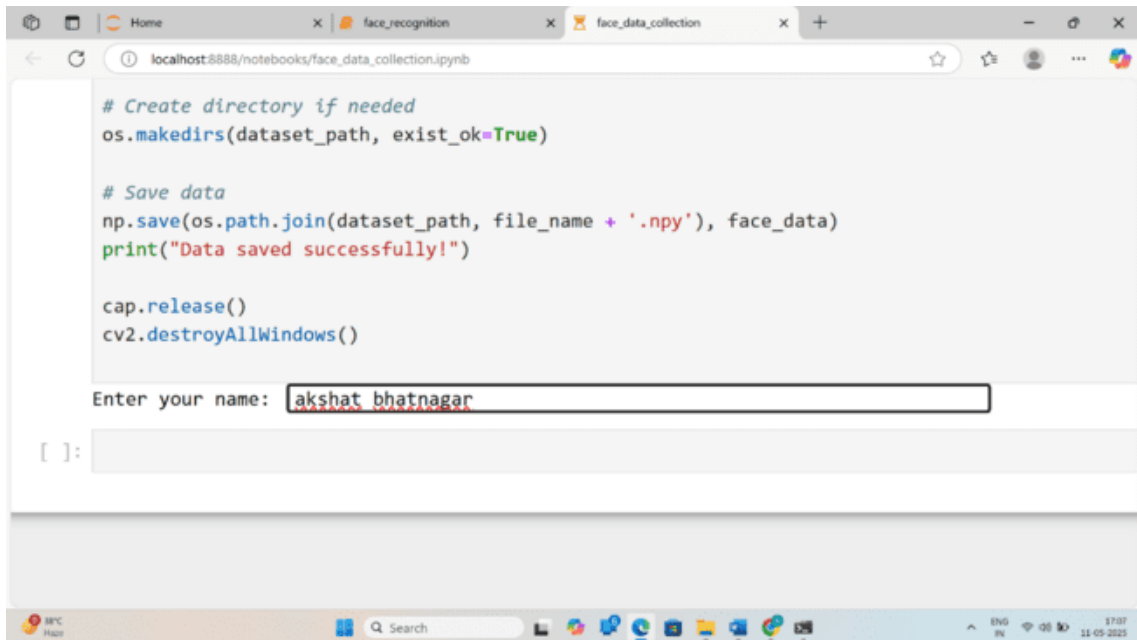
### Output Screens

This section displays all relevant user interfaces and output screens generated during the execution of the Face Recognition system. Since the system is primarily CLI-based with GUI output from OpenCV, outputs are shown via real-time windows and console messages.

---

#### 8.1 Console Output – System Start

- Description: Displayed when the program begins execution.
- Output: Shows status messages indicating that known face images are being loaded.



The screenshot shows a web browser window displaying a Jupyter Notebook. The notebook has two tabs: 'face\_recognition' and 'face\_data\_collection'. The active tab is 'face\_data\_collection', which contains the following Python code:

```
# Create directory if needed
os.makedirs(dataset_path, exist_ok=True)

# Save data
np.save(os.path.join(dataset_path, file_name + '.npy'), face_data)
print("Data saved successfully!")

cap.release()
cv2.destroyAllWindows()
```

Below the code, there is a text input field with the label "Enter your name:". The field contains the text "akshat bhatnagar". Below the input field is a prompt "[ ]:". The browser's address bar shows "localhost:8888/notebooks/face\_data\_collection.ipynb". The Windows taskbar is visible at the bottom of the screen.

#### System Start

Loading known face images...

Encoding faces...

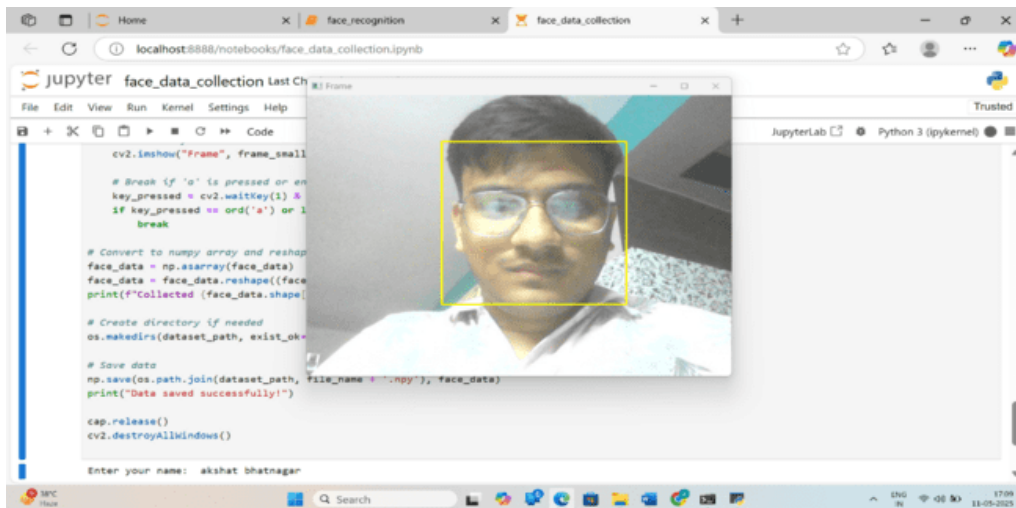
Starting video stream...

Press 'q' to quit.

---

## 8.2 GUI Output – Initial Webcam Feed

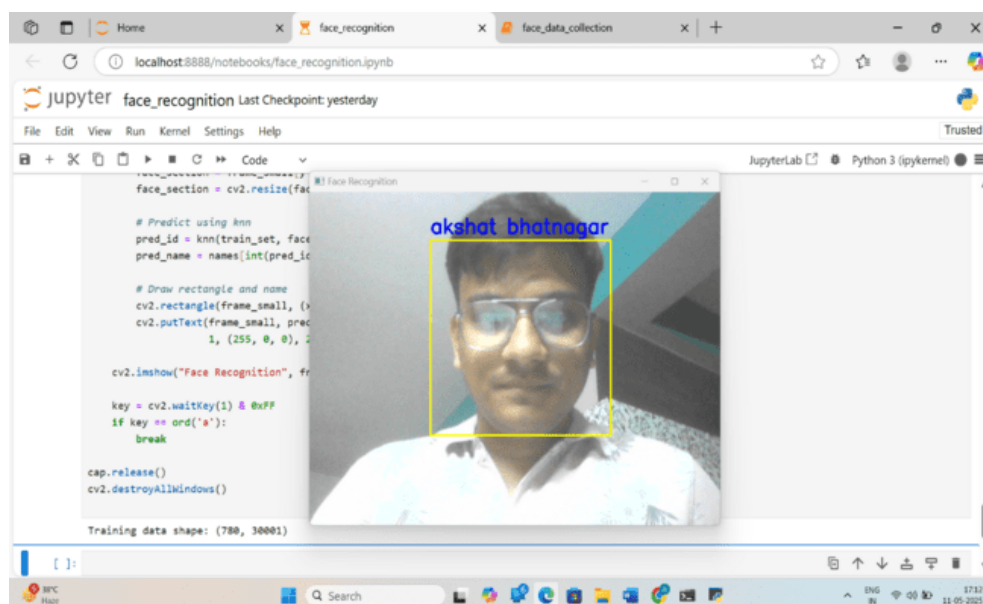
- Screen: OpenCV window displaying live webcam footage.
- Description: No face detected yet.



### Initial Webcam Feed

## 8.3 GUI Output – Known Face Detected

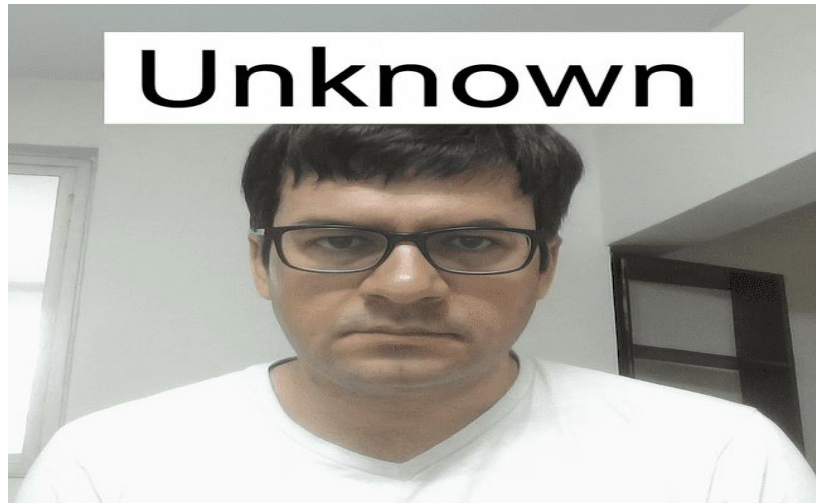
- Screen: Detected face with name label (e.g., “Akshat”) under the bounding box.
- Description: A known face from the images/ folder is recognized in real time.



- **Known Face Detected**

8.4 GUI Output – Unknown Face Detected

- Screen: Unknown person detected and labeled as “Unknown”.
- Description: Face detected is not part of the training dataset.



**Unknown Face Detected**

---

.

8.6 Console Output – System Exit

- Description: When the user presses 'q' to quit the application.

## CHAPTER 9

### Conclusion

- 1) The Face Recognition system offers a **real-time solution** for identifying individuals via **webcam input**.
- 2) Developed using **Python** and key libraries: `face_recognition`, `OpenCV`, and `NumPy`.
- 3) Accurately **detects, encodes, and recognizes faces** by comparing them with a known dataset.
- 4) Performs reliably under **standard lighting conditions**.
- 5) Provides **instant visual feedback** using bounding boxes and labels on the video feed.
- 6) **Modular design** allows for easy addition of new faces to the system.
- 7) Can be adapted for various applications such as:
  - **Biometric login systems**
  - **Attendance management**
  - **Security surveillance**
- 8) Demonstrates how **computer vision** and **machine learning** can be used to solve real-world identity verification problems in a **lightweight, cost-effective** manner.



## CHAPTER 10

### Further Enhancements / Recommendations

#### GUI integration:

Implementing a **graphical user interface** using frameworks like **Tkinter** or **PyQt** would improve accessibility and usability, especially for non-technical users.

#### Database Integration:

Store facial data and logs in a **relational database** such as **SQLite** or **MySQL** to enable better data management, scalability, and analytics.

#### Liveness Detection:

Integrate **liveness detection** techniques to differentiate between real faces and spoofing attempts (e.g., printed photos or videos), enhancing security.

#### Advanced Deep Learning Models:

Use **custom CNN architectures** or pre-trained models via **TensorFlow** or **PyTorch** to increase recognition **accuracy** under challenging conditions (e.g., varied lighting, occlusion).

#### Face Mask Detection & Demographics:

Add modules for **face mask detection**, **age estimation**, and **gender classification** to extend the system's capabilities.

#### Cloud Integration:

Enable **cloud-based synchronization** of facial datasets and logs to support centralized management and remote access.

#### Edge & IoT Deployment:

Deploy the system on **edge devices** (e.g., Raspberry Pi, NVIDIA Jetson) or integrate with **IoT systems** for applications like smart surveillance, automation, or access control.

#### **Liveness Detection:**

Integrate **liveness detection** techniques to differentiate between real faces and spoofing attempts (e.g., printed photos or videos), enhancing security.

#### **Database Integration:**

Store facial data and logs in a **relational database** such as **SQLite** or **MySQL** to enable better data management, scalability, and analytics.

#### **Cloud Integration:**

Enable **cloud-based synchronization** of facial datasets and logs to support centralized management and remote access

## CHAPTER 11

### References / Bibliography

- Ageitgey, Adam. *face\_recognition* GitHub repository.  
[https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)
- OpenCV Documentation. <https://docs.opencv.org/>
- NumPy Documentation. <https://numpy.org/doc/>
- Akshat Bhatnagar, *Face Recognition* Project. GitHub.  
<https://github.com/Akshatbhatnagar908/Face-Recognition->
- Python Software Foundation. <https://www.python.org/doc/>
- Real Python Tutorials. <https://realpython.com/>

## CHAPTER 12

### Appendices

#### Appendix A: Code Snippet – Face Encoding

```
import face_recognition
```

```
# Load an image of the person
```

```
image = face_recognition.load_image_file("images/person1.jpg")
```

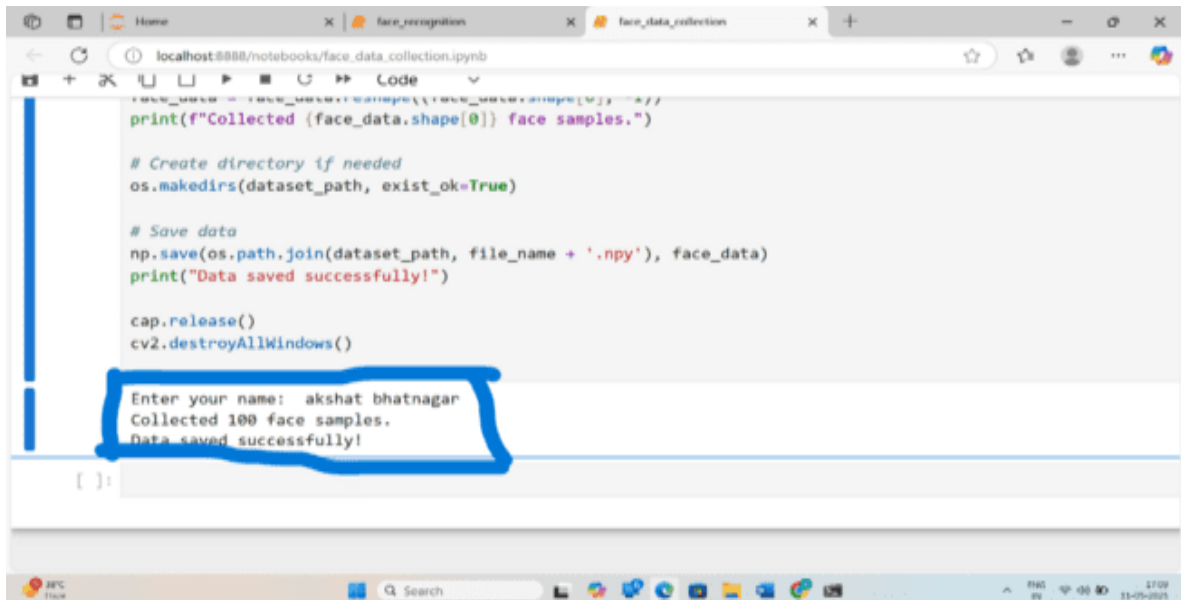
```
# Generate face encoding
```

```
encoding = face_recognition.face_encodings(image)[0]
```

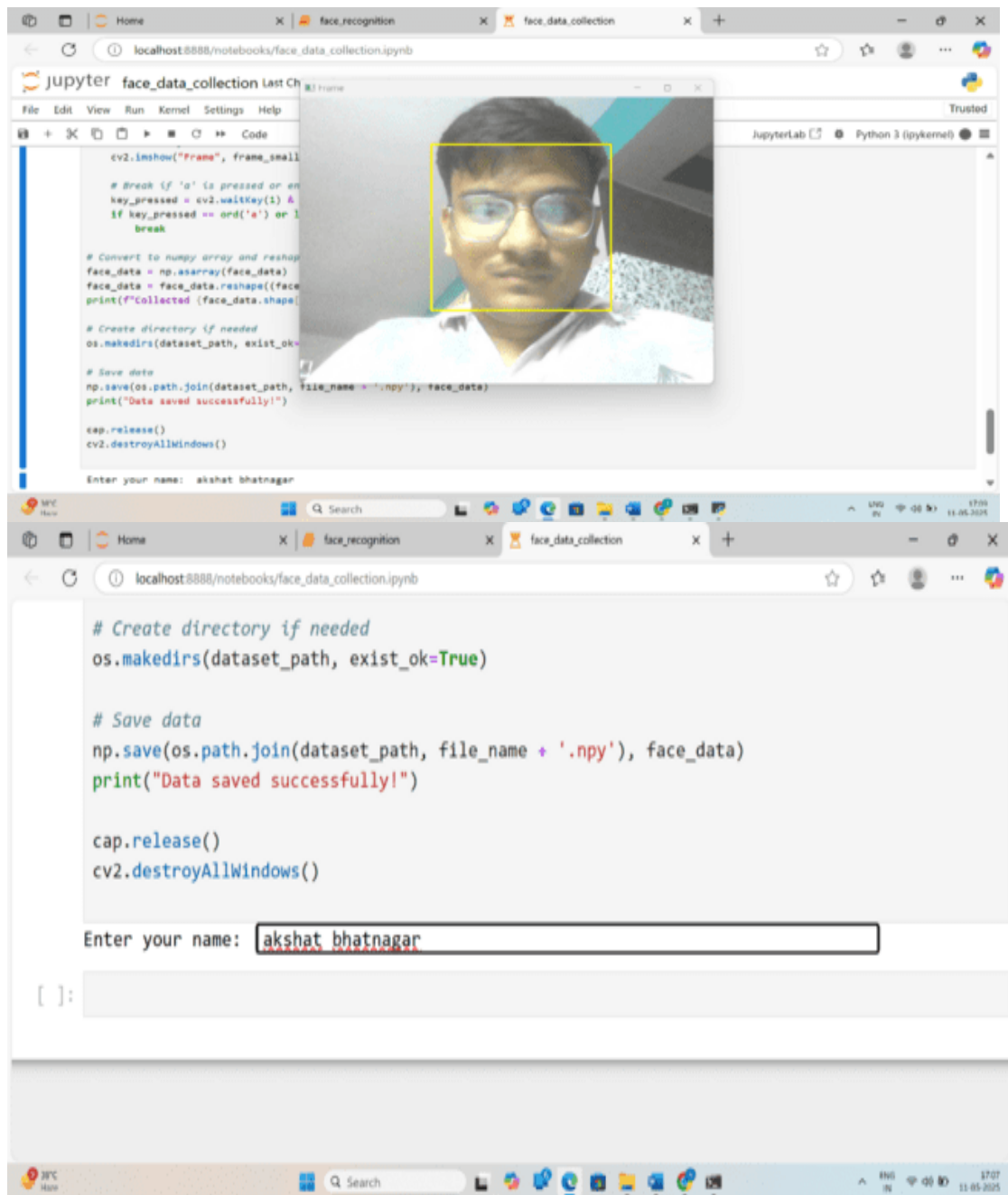
```
# Append encoding and name to respective lists
```

```
known_face_encodings.append(encoding)
```

```
known_face_names.append("Person 1")
```



### Face Encoding



## Face Encoding

### Appendix B: Sample Output

Visual Output:

- Real-time webcam feed with bounding boxes drawn around detected faces.

- Labels displayed above faces such as "John" or "Unknown".

Console Output:

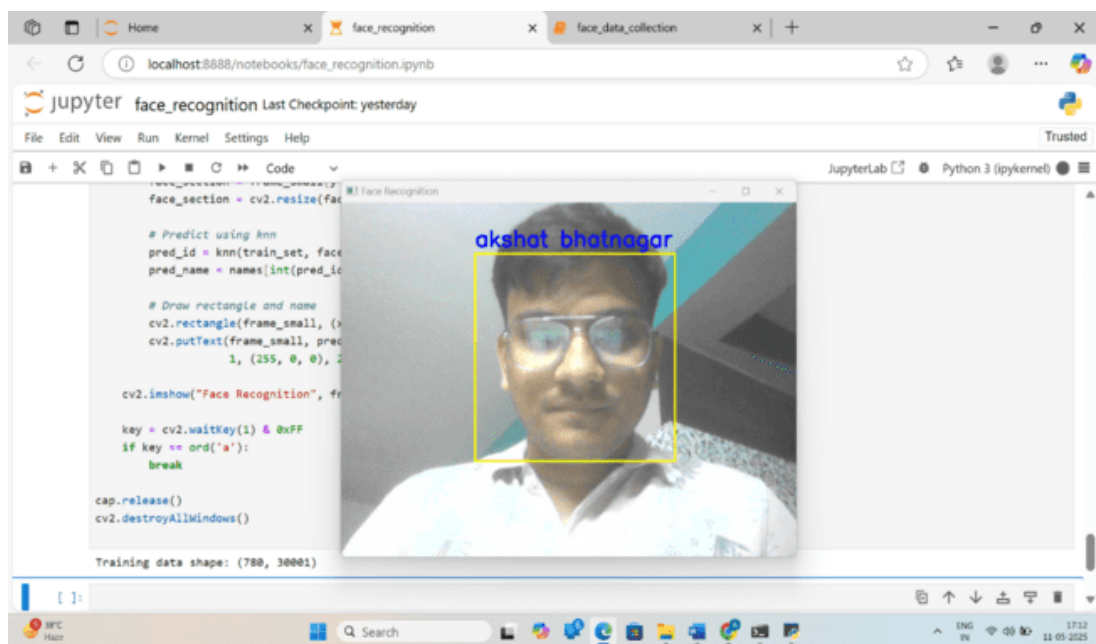
Loading known faces...

Starting webcam...

Face recognized: John

Face recognized: Unknown

Exiting program...



## Sample Output