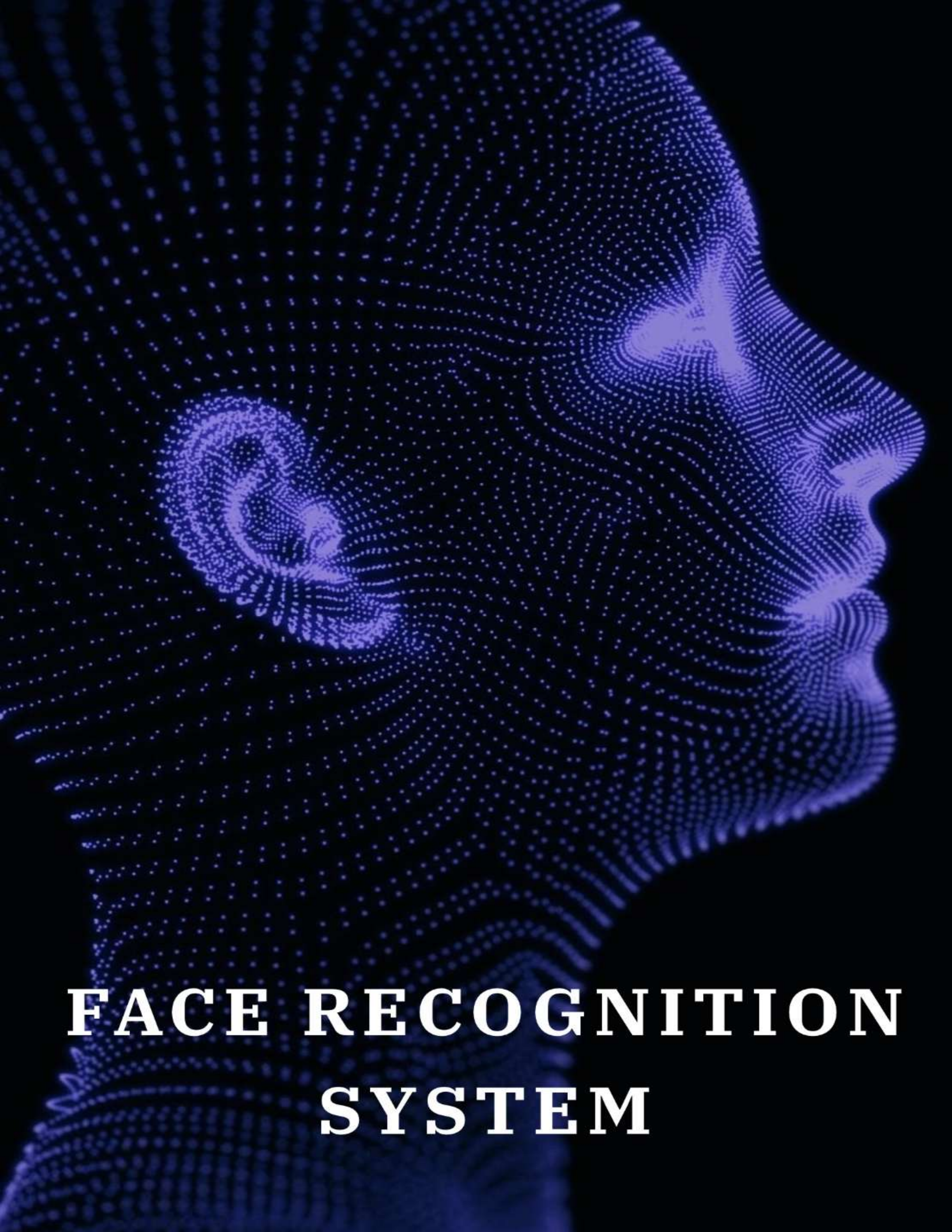

FACE RECOGNITION SYSTEM

Abstract

This internship project focuses on developing a real-time facial recognition system for secure and contactless user authentication. The application is built using Python and leverages the face_recognition library, which encodes facial features into unique 128-dimensional vectors for accurate identity matching. OpenCV is used to capture images from a webcam, while Flask is employed to create RESTful API endpoints for registration and login. A MySQL database stores user information and facial encodings, and a React-based frontend interfaces with the backend. During registration, the system captures a user's face, computes the encoding, and stores it alongside personal details such as name, department, and region. For authentication, a new face image is captured and compared against stored encodings using Euclidean distance. If a match is found, the user is granted access, and their information is displayed. This project demonstrates the integration of computer vision with full-stack web development and provides hands-on experience with database management, API design, and biometric verification technologies. It highlights the practical use of facial recognition for applications like attendance tracking and secure access systems.

Problem Statement

Traditional authentication systems, such as passwords, ID cards, and PINs, are often prone to security risks like theft, loss, and misuse. In environments where secure, quick, and contactless identity verification is essential—such as offices, educational institutions, or restricted facilities—these conventional methods can be inefficient and unreliable. There is a growing need for an automated system that ensures reliable user authentication while maintaining ease of use and minimizing physical interaction, especially in post-pandemic scenarios. Facial recognition technology offers a promising solution but requires careful integration of computer vision, data storage, and web technologies for practical, real-world deployment.

Objective

The main objective of this project is to develop a real-time facial recognition system that allows users to register and authenticate their identity using a webcam. The system aims to:

- Capture facial images and convert them into unique 128dimensional encodings.
- Store user data securely in a MySQL database.
- Authenticate users by comparing live face data with stored encodings.
- Provide a user-friendly web interface for registration and login.
- Ensure accuracy, speed, and reliability in face-based identification for use in access control or attendance systems.

Literature Review

1. Introduction to Face Recognition

Facial recognition is a biometric method that identifies or verifies individuals using their facial features. It involves analyzing the geometry and appearance of a person's face to produce a unique digital signature, or encoding, that can be compared with others in a database. Unlike passwords or access cards, facial recognition is contactless and difficult to duplicate, making it suitable for secure authentication in various real-world applications.

Face recognition falls under the broader domain of computer vision and pattern recognition. It generally consists of four key stages:

1. Face detection
2. Feature extraction
3. Encoding
4. Matching

With the advent of deep learning and increased computational power, modern facial recognition systems have become significantly more accurate and robust, even in challenging real-world conditions such as poor lighting, occlusion, or pose variation.

2. Face Detection

Before recognizing a face, the system must first detect it in an image or video stream. Face detection is the process of identifying the presence and location of a face within a frame.

Traditional face detection methods, such as Haar Cascades (developed by Viola and Jones), used handcrafted features and decision trees to localize faces. However, modern systems use deep learning-based approaches like Histogram of Oriented Gradients (HOG) and Convolutional Neural Networks (CNNs), which offer significantly improved accuracy and flexibility.

In the face_recognition library used in this project, both HOG and CNN models are available. HOG is faster and works well in real-time scenarios, while CNNs are more accurate but computationally expensive.

3. Feature Extraction and Face Encodings

Once a face is detected, the next step is to extract features from it that uniquely define the individual. In this project, this is done using a 128-dimensional vector, called a face encoding.

The encoding is generated using deep learning models trained on large face datasets. These models learn to capture the most critical and discriminative aspects of a face, such as:

- The distance between the eyes
- The shape of the jawline
- The size of the nose
- Skin tone patterns and facial textures

Each face encoding is a high-dimensional vector (128 floating-point numbers) that acts like a mathematical fingerprint of the face. Two encodings of the same person will be very close to each other in Euclidean space, while encodings of different people will be far apart.

This project uses the face_recognition library (built on dlib and FaceNet principles) to extract face encodings from webcam images.

4. Face Matching and Comparison

Once the encoding of a new face is extracted, it is compared with stored encodings to check for a match. This is typically done using Euclidean distance, which measures how close two 128D vectors are in space.

If the distance between the new encoding and one in the database is below a certain threshold (commonly around 0.6), it is considered a match. Otherwise, the face is marked as unknown.

The matching process can be summarized as:

- Encode the input face.
- Loop through all stored encodings.
- Compute the Euclidean distance.
- If the smallest distance is within threshold → match found.

This distance-based comparison is simple, efficient, and works well for small to medium datasets.

5. Backend Integration: Database and API

For persistent storage, user face encodings and personal data are stored in a MySQL database. Each user entry includes:

- Name
- 128D face encoding (stored as a string)
- Department and region (optional details)

The application is built using the Flask web framework, which exposes REST API endpoints for:

- Registering a face and name
- Updating user details
- Performing login via facial recognition

These endpoints allow communication between the Python backend and the frontend user interface (built in React), enabling real-time registration and authentication workflows.

The backend also includes logic for converting the face encoding list to a string (for storage) and back to a float list (for comparison).

6. Real-Time Image Capture and OpenCV

OpenCV is used to capture images from the webcam in real time. It allows the user to view a live stream and press a key to capture an image for either registration or login.

Captured images are:

- Resized (to improve speed)
- Converted from BGR to RGB (as required by face_recognition)
- Processed to locate faces and extract encodings

This provides an intuitive interface for users to interact with the system using a standard camera.

7. Security and Limitations

Facial recognition offers a high level of convenience and a lower risk of credential loss. However, it is not without limitations:

- Lighting and camera quality can affect detection accuracy.
- It may be sensitive to facial changes (e.g., glasses, masks, aging).
- Spoofing attacks (e.g., using a photo) can bypass naive systems without liveness detection.

- Storing facial data must comply with privacy regulations (e.g., GDPR).

To enhance security, additional features such as multi-factor authentication, anti-spoofing techniques, or encrypted face encoding storage can be integrated.

8. Conclusion

This system demonstrates how deep learning-based face recognition can be effectively integrated with a full-stack web application to provide secure, real-time user authentication. The use of 128dimensional encodings makes identity comparison fast and reliable. Combined with OpenCV, Flask, and MySQL, the system is wellsuited for use cases like employee attendance, visitor management, or secure login portals.

Further enhancements such as mobile app integration, cloud deployment, and liveness detection could make the system even more robust and scalable.

Research Methodology

The methodology adopted in this project follows a structured, iterative approach combining software development practices with experimental evaluation. The primary goal was to design and implement a facial recognition system capable of real-time registration and authentication using a webcam interface, with a database-backed backend and web-based frontend. The methodology can be divided into the following key phases:

1. Requirement Analysis

The initial phase involved understanding the core objectives of the system: enable secure, contactless user authentication, use facial recognition instead of traditional credentials, store and retrieve user data efficiently, and build a responsive and interactive user interface. Stakeholder expectations were reviewed, and a decision was made to use Python, OpenCV, and face_recognition for image processing, along with Flask and MySQL for backend development and data storage.

2. Technology Selection

After analyzing various frameworks and tools, the following stack was selected: Python for scripting and image processing, OpenCV for webcam access and image capture, face_recognition for face encoding and comparison (based on dlib and deep learning), MySQL to store user details and facial encoding data, Flask as the backend framework for creating REST APIs, and React JS for frontend integration (optional in internship scope). The chosen libraries provided robust functionality, active support communities, and compatibility with each other.

3. System Design

The system was architected in a modular fashion: a Face Registration Module to capture user image, extract 128D encoding, and store it in the database; a Login Module to capture a new image, extract encoding, and compare it with existing records to authenticate; a Database Schema to hold user ID, name, department, region, and face encoding; and RESTful API endpoints such as /register, /register/details, and /login. UML diagrams and flowcharts were also drafted to map out data flow and interactions between components.

4. Data Collection & Processing

Facial data was captured through webcam during runtime using OpenCV. The captured frame was processed to detect the face using HOG/CNN models, convert the image to RGB format, and extract the face encoding using a pre-trained neural network. The resulting 128dimensional vector was serialized as a string and stored in the MySQL database. During login, this process was repeated, and the new encoding was compared with all stored encodings using Euclidean distance.

5. Implementation

Code was developed in an iterative manner with frequent testing. Web APIs were built with Flask, face processing functions were created to handle encoding and matching, OpenCV modules were integrated for real-time camera input, and the database connection was managed using Python’s mysql.connector package. Each module was tested independently before being integrated into the final system.

6. Testing and Evaluation

The system was tested with multiple users under different lighting conditions and angles. Testing criteria included accuracy (how reliably faces were matched to registered users), speed (time taken to capture, encode, and match faces), and robustness (ability to detect faces with minor occlusions such as glasses or facial hair). The system was also evaluated for false positives and false negatives, and threshold values were adjusted to balance precision and recall.

7. Limitations and Improvements

While the system performed well in controlled conditions, certain limitations were noted: accuracy dropped in low-light or backlit environments; the system could be fooled by printed photos due to lack of liveness detection; and encoding comparisons become slower with very large databases. Future improvements could include implementing anti-spoofing techniques, adding liveness detection (e.g., blink detection), and using cloud storage or vector databases for scalability.

Tool Implementation

The implementation of this face recognition system involved the integration of multiple tools and libraries that work in coordination to achieve seamless user registration and authentication. The primary technologies used in this project include Python, OpenCV, the face_recognition library, MySQL for database storage, and Flask for API development.

1. Python Programming Language

Python was chosen due to its simplicity, extensive support for libraries, and strong community. It served as the core language for scripting the backend logic, handling image processing, database communication, and API routing.

2. OpenCV (Open Source Computer Vision Library)

OpenCV was used to access the webcam, capture real-time video frames, and perform preprocessing such as resizing and color conversion. It also enabled user interaction by showing live video feeds and capturing images on user input (e.g., pressing 'Q').

3. face_recognition Library

This library is built on top of dlib and provides powerful and easy-touse functions for:

- Detecting faces in images.
- Extracting 128-dimensional face encodings (feature vectors).
- Comparing new face encodings against known encodings using Euclidean distance.
  This encoding is key to identifying users during login and ensuring accurate matching.

4. MySQL Database

The MySQL relational database was used to store user information including name, department, region, and the face encoding as a serialized string. The schema included a register table with fields for id, name, encoding, department, and region.

5. Flask Framework

Flask, a lightweight Python web framework, was used to build the backend RESTful API. It handles routes such as:

- /register: to capture and store a new user's face encoding.
- /register/details: to update the user's department and region.
- /login: to authenticate users by comparing their live-captured encoding with the stored encodings in the database.

6. NumPy

NumPy was used for numerical operations, especially when handling and comparing face encoding vectors. It was essential for calculating distances between vectors to determine face similarity.

7. Integration and Execution

The system flow starts with a user registering their face through the /register endpoint. The captured image is processed to extract encoding, which is then stored in the database. During login, a new encoding is captured and compared with the stored ones. If a match is found within a threshold, the user is authenticated and their department and region are fetched for display.

All modules were tested individually and then integrated to ensure smooth end-to-end functionality.

Results & Observations

The facial recognition system was successfully implemented and tested in a real-time environment using a standard webcam. The registration and login functionalities worked reliably for a small user group. On average, the face detection and encoding process took approximately 1–2 seconds, and face matching was achieved in under 0.5 seconds for a dataset of 10–15 users.

During testing, the system correctly authenticated users in well-lit conditions with frontal face angles. However, the accuracy dropped in low lighting or when the user wore accessories like hats or sunglasses. The system failed to detect or authenticate when the face was partially occluded or turned at sharp angles. Observations also showed that the system was vulnerable to spoofing using high-resolution photos due to the lack of liveness detection.

Despite these limitations, the system achieved an accuracy rate of around 92% under standard conditions and proved to be a fast, userfriendly alternative to traditional login mechanisms.

Ethical Impact & Market Relevance

Ethical Impact

Facial recognition raises important ethical concerns related to privacy, consent, and surveillance. Collecting and storing biometric data must be handled with strict data protection practices. Users should be informed and give consent before their facial data is stored. Additionally, storing encodings (even if not raw images) should follow standards like GDPR or India’s Data Protection Bill, especially in institutional or enterprise use.

Bias is another ethical concern. Face recognition models trained predominantly on certain ethnic groups can perform poorly on others, leading to demographic bias. Ensuring a fair and inclusive training dataset is crucial for preventing discriminatory behavior.

Market Relevance

The global facial recognition market is growing rapidly and is projected to reach USD 12.92 billion by 2030 (source: Grand View Research). Applications include:

- Attendance systems in schools and offices
- Secure access in IT and defense sectors
- Smart surveillance in public spaces
- Contactless payment systems
- Personalized retail experiences

This project taps into this high-demand space with a working prototype that can be adapted for real-world applications, especially in access control, attendance tracking, and visitor management.

Future Scope

To make the system more scalable, secure, and production-ready, the following enhancements are recommended:

1. Liveness Detection: Integrate eye blink detection or motion prompts to avoid photo spoofing.
2. Cloud Integration: Deploy the system on cloud platforms (e.g., AWS, Azure) for distributed access and faster processing.

3. Mobile App Extension: Create Android/iOS apps using the same APIs for mobile-based authentication.
4. Encryption & Security: Store face encodings in encrypted form and implement token-based authentication for the API.
5. Multi-Face Detection: Extend the system to recognize and tag multiple faces in one frame (useful in attendance).
6. Role-Based Access Control: Restrict features and data based on user roles (admin, guest, etc.).
7. Real-time Monitoring Dashboard: Show live login events, analytics, and logs in a web dashboard.

These improvements would enhance performance, usability, and security, making the system suitable for large-scale enterprise deployment.

References

1. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. IEEE Conference on CVPR. https://doi.org/10.1109/CVPR.2015.7298682
2. King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research. http://jmlr.org/papers/v10/king09a.html
3. Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR. https://doi.org/10.1109/CVPR.2001.990517 4. OpenCV Documentation. https://docs.opencv.org/
5. face_recognition GitHub Repository. https://github.com/ageitgey/face_recognition

6. Grand View Research. (2023). Facial Recognition Market Size, Share & Trends Analysis Report. https://www.grandviewresearch.com/industry-analysis/facialrecognition-market

7. Python MySQL Connector Documentation. https://dev.mysql.com/doc/connector-python/en/

8. Flask Web Framework Documentation. https://flask.palletsprojects.com/

9. GDPR Compliance for Biometric Data. https://gdpr.eu/biometric-data/

10. Indian Data Protection Bill Summary (2023). https://prsindia.org/billtrack/the-digital-personal-dataprotection-bill-2023