

Face Recognition System using Python and OpenCV

Nishant Kumar¹

¹M.Tech Scholar, Dept. of Computer Science Engineering & Information Technology,
Institute of Engineering and Technology, Mangalayatan University, Beswan, Aligarh, India

Abstract - In an age where security and automation are increasingly prioritized, face recognition systems provide an effective and non-intrusive method for identity verification. This paper presents a Face Recognition System built using Python, OpenCV, and the Local Binary Pattern Histogram (LBPH) classifier, with a GUI developed in Tkinter and a MySQL backend. The system covers facial data acquisition, model training, real-time recognition, and database-based user management. Testing on a standard laptop webcam yielded a recognition accuracy of approximately 95% under well-lit, frontal conditions, with recognition time under one second. The system demonstrates high reliability for attendance management and access control in academic or office environments.

Key Words: Face Recognition, LBPH Algorithm, Haar Cascade, OpenCV, Python, Tkinter GUI, MySQL Database, Real-Time Detection

1. INTRODUCTION

Biometric authentication has gained widespread adoption due to its potential to deliver secure and efficient personal identification without physical tokens. Among the various biometric modalities, face recognition stands out as non-intrusive, contactless, and increasingly practical given the ubiquity of cameras in smartphones, laptops, and surveillance systems [1]. Unlike fingerprint or iris scanning, face recognition requires no physical contact, making it ideal for situations where hygiene or ease of use is a concern.

Conventional attendance and identity verification methods such as manual roll calls, RFID cards, and PINs are often inefficient and susceptible to fraud. Cards can be lost, PINs can be shared, and manual methods consume significant time, especially in large-population environments such as schools, universities, and corporate offices. Biometric methods eliminate proxy attendance and reduce administrative overhead significantly.

The motivation behind this work stems from the need for an automated, accessible, and deployable system that integrates computer vision, a user-friendly GUI, and database management into a cohesive application. This paper describes the design, implementation, and evaluation of a complete face recognition pipeline using Python and OpenCV that operates entirely offline, requires no specialised hardware beyond a standard webcam, and is suitable for deployment in resource-constrained environments.

1.1 Problem Statement

Existing face recognition solutions either demand high-end GPU hardware (deep learning models) or lack integration with complete workflows including registration, GUI, and persistent storage. There is a clear need for a modular, and accessible system that can be deployed on standard computers without cloud dependency. The proposed system addresses this gap using classical computer vision combined with a structured application architecture.

1.2 Objectives

The key objectives of this project are: (1) Data Acquisition — design a module to capture diverse facial images via webcam or video files; (2) Classifier Training — implement the LBPH algorithm to train per-user recognition models; (3) Real-Time Recognition — identify faces from a live webcam feed with latency under one second; (4) GUI Development — build an intuitive Tkinter-based interface accessible to non-technical users; (5) Database Integration — store and manage user profiles in MySQL; (6) Access Control — implement admin login authentication to protect the system.

1.3 Scope and Limitations

The system is designed for controlled indoor environments such as classrooms, laboratories, and small offices. It supports single-face recognition per session and requires an initial enrolment phase. Recognition accuracy may degrade under poor lighting, significant head rotation, or partial face occlusion. The system does not currently support mask detection, multi-face simultaneous recognition, or cloud-based deployment, though these are identified as future enhancements.

1.4 Technology Stack

Python 3.9 serves as the primary language for its extensive library ecosystem. OpenCV 4.x provides face detection via the Haar Cascade Classifier and recognition via the LBPH recognizer. Tkinter is used for cross-platform desktop GUI development. MySQL 8.0 with mysql-connector-python handles persistent storage. The LBPH algorithm was selected for its robustness against monotonic illumination changes, its support for incremental training, and its built-in availability within OpenCV without requiring separate model files or GPU resources.

2. LITERATURE REVIEW

Face recognition research has evolved through several distinct phases. Early systems relied on geometric methods that measured distances between facial landmarks such as the eyes, nose tip, and mouth corners. While intuitive, these approaches proved sensitive to changes in pose, lighting, and expression.

The 1990s and early 2000s saw the rise of appearance-based statistical methods. Turk and Pentland introduced Principal Component Analysis (PCA)-based Eigenfaces, which represented faces as linear combinations of basis vectors derived from training data. Belhumeur et al. extended this with Linear Discriminant Analysis (LDA) in the Fisherfaces method, which maximised inter-class separation while minimising intra-class variation. Both methods represented significant milestones but remained limited by their linear assumptions.

The deep learning era brought transformative improvements. Taigman et al. (DeepFace, Facebook) achieved near-human accuracy on the LFW benchmark using a 9-layer deep neural network with 3D alignment [2]. Schroff et al. (FaceNet, Google) introduced a triplet-loss-based approach that maps faces into a compact Euclidean embedding space, enabling direct distance-based comparison [3]. VGG-Face (Parkhi et al., Oxford) demonstrated that very deep networks trained on large datasets could generalise robustly across diverse conditions.

Despite these advances, deep learning models require GPU acceleration, datasets of tens of thousands of images, and substantial engineering effort to deploy. For academic prototypes and resource-constrained applications, the Local Binary Patterns Histogram (LBPH) method remains a practical and well-validated choice [4]. LBPH encodes local micro-texture by comparing each pixel to its P neighbours on a circle of radius R, producing a binary pattern. These patterns are accumulated into spatial histograms and compared using chi-square distance or histogram intersection for recognition.

2.1 Comparison of Approaches

Table 1: Comparison of Face Recognition Approaches

Method	Accuracy	Hardware Req.	Complexity
Eigenfaces (PCA)	Moderate	Low	Low
Fisherfaces (LDA)	Good	Low	Medium

LBPH	Good	Low (CPU)	Low
DeepFace / FaceNet	Very High	GPU needed	High
VGG-Face	Very High	GPU needed	High

2.2 Identified Gaps and Proposed Solution

Key gaps include: (1) Deep-learning solutions demand GPU hardware unavailable in typical lab settings; (2) Most research prototypes lack integrated GUIs and database backends; (3) Real-world deployment needs offline operation without cloud API dependencies; (4) Hardcoded or undocumented enrolment pipelines limit accessibility for non-expert users. The proposed system addresses all these gaps by combining LBPH recognition with a modular Python application, Tkinter GUI, and MySQL database, operating entirely offline on commodity hardware.

3. SYSTEM DESIGN & ARCHITECTURE

The Face Recognition System is organised around a six-module architecture that separates concerns cleanly across authentication, user interface, data acquisition, model training, recognition, and database management. This design promotes independent testing and future replacement of individual components without impacting the rest of the application.

Figure 1 shows the block diagram illustrating data flow: raw webcam frames are captured and pre-processed, faces are detected and cropped, features are extracted via LBPH, and the resulting model is persisted for recognition. User data is stored in MySQL throughout the pipeline.

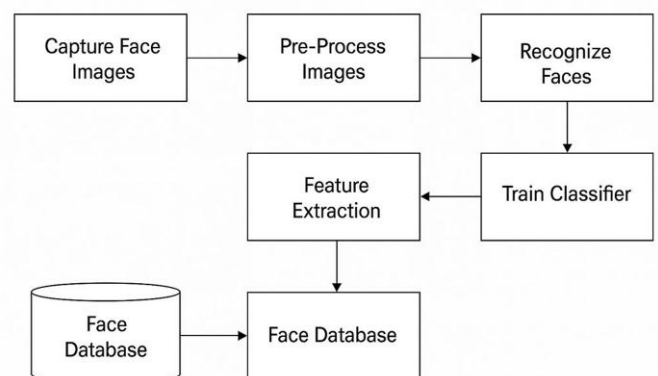


Fig 1: System Block Diagram

3.1 Authentication Module (login_page.py)

The login module presents a minimal Tkinter window requesting admin credentials. Credentials are currently validated against hardcoded values (username: admin, password: admin123). On success, the subprocess module spawns the main GUI process. This module prevents unauthorised access to sensitive registration and recognition features. Future work will migrate credential storage to a hashed database record.

3.2 Data Acquisition Module (create_dataset.py)

The module initialises the Haar Cascade Classifier and opens the webcam via cv2.VideoCapture(0). For each frame, detected face ROIs are converted to grayscale and saved as JPEG images in a user-specific directory (./data/<username>/). The process terminates after 300 images are captured, ensuring adequate dataset variety for LBPH training. An alternative path, take_video(), allows frame extraction from pre-recorded video files, supporting offline enrolment scenarios.

3.3 Classifier Training Module (create_classifier.py)

All face images in the data directory are loaded, converted to grayscale, and assigned numeric labels corresponding to each user folder. The LBPH recognizer is created and trained using recognizer.train(faces, labels). The radius (R=1), number of neighbours (P=8), grid dimensions (8x8 cells), and histogram comparison method are kept at OpenCV defaults, which have been empirically validated for this class of applications. The trained model is saved as an XML file using recognizer.save().

3.4 Recognition Module (detector.py)

The recognition engine loads the classifier XML and Haar Cascade, then enters a frame-processing loop. Each frame is converted to grayscale and scanned for face regions using detectMultiScale() with scaleFactor=1.3 and minNeighbors=5. Detected ROIs are passed to recognizer.predict(), returning a (label, confidence) pair. Confidence scores below 50 indicate a match; the user's name is drawn in green above a green bounding box. Unknown faces receive a red bounding box. A five-second session timeout prevents indefinite loops when no face is present.

3.5 Database Module (db_config.py)

A centralised get_connection() function establishes a mysql.connector connection to the face_recognition database. The users table stores id (PK), name, age, gender, height, location, and project fields. All GUI registration actions write to this table. Separating the connection logic into a dedicated module simplifies testing and credential rotation without modifying application logic.

3.6 System Workflow

The end-to-end workflow is: (1) Admin authenticates via login page; (2) New user details are entered and registered in MySQL; (3) Face images are captured and stored; (4) LBPH classifier is trained on captured images; (5) Real-time recognition is initiated via webcam; (6) Recognition result displayed with bounding box and name label; (7) Check-in status logged and confirmation shown via message box.

4. IMPLEMENTATION & RESULTS

All modules were implemented in Python 3.9 on Windows 10, using OpenCV 4.x, Tkinter, and mysql-connector-python. The complete source code spans seven Python files and is structured for modularity and reusability. The GUI flow begins at the Login Page (Fig. 2), proceeds to the Main Dashboard (Fig. 3), then to face capture (Fig. 4), model training confirmation (Fig. 5), and real-time recognition (Fig. 6). Message box confirmations (Fig. 7) provide user feedback, while Fig. 8 shows the MySQL database records.

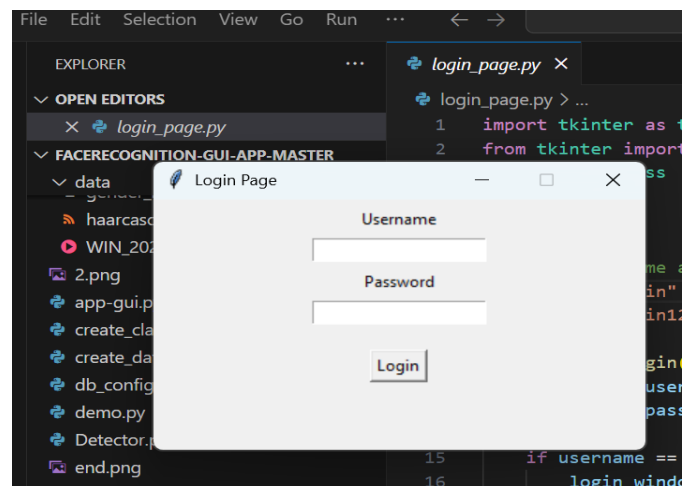


Fig 2: Login Page – Admin Authentication

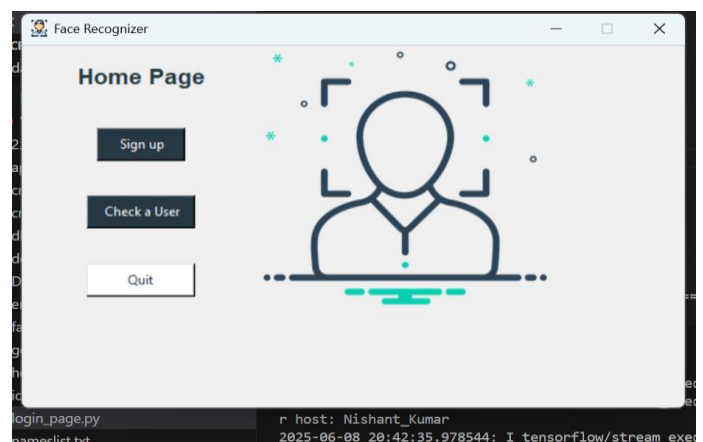


Fig 3: Main Dashboard – Sign Up, Check User, Quit



Fig 4: Face Capture – Haar Cascade Bounding Box

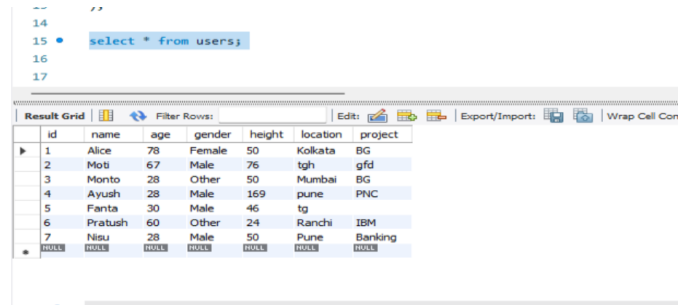


Fig 8: MySQL Database – Registered User Records

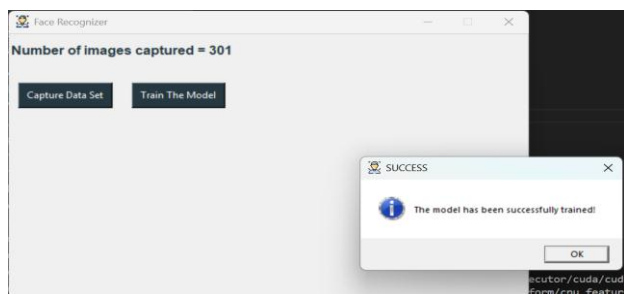


Fig 5: Model Training – Success Confirmation (301 images)

4.1 Database Schema

Table 2: Database Schema – users table

Field	Type	Description
id	INT (PK)	Unique User ID
name	VARCHAR(100)	Full Name
age	INT	User Age
gender	VARCHAR(20)	Gender
height	VARCHAR(10)	Height (cm)
location	VARCHAR(100)	City/Location
project	VARCHAR(100)	Project Name

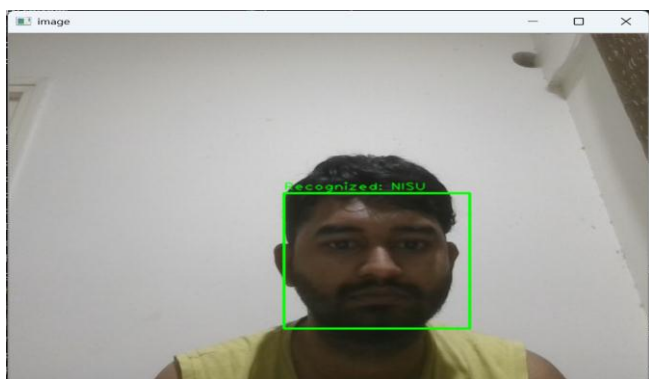


Fig 6: Real-Time Recognition – Known User (Green Box)

4.2 Folder Structure

The project directory is organised as follows: root-level Python scripts (app-gui.py, login_page.py, create_dataset.py, create_classifier.py, detector.py, predict.py, db_config.py); a data/ subdirectory containing haarcascade_frontalface_default.xml, a classifiers/ folder for trained XML models, and per-user subdirectories storing captured face images; and a face_recognition.sql schema file for database initialization.

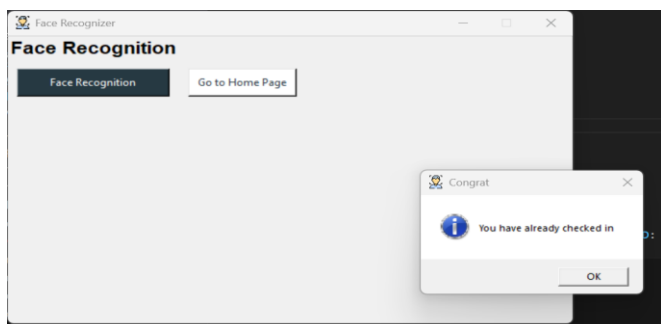


Fig 7: Check-In Confirmation Message Box

5. TESTING & EVALUATION

Testing was conducted at three levels: (1) Unit Testing — validating individual module functionality in isolation; (2) Integration Testing — verifying seamless end-to-end operation from login to recognition; (3) User Acceptance Testing — assessing usability and GUI clarity with three test users. The test environment comprised Windows 10, Python

3.9, MySQL 8.0, and a standard integrated laptop webcam operating at 720p resolution with 4 GB RAM.

5.1 Unit Test Results

Table 3: Unit Test Summary

Module	Test Case	Expected Result	Status
create_dataset.py	Capture & save images	300+ images in /data/<name>/	✓ Pass
create_classifier.py	Train & save model	XML model file generated	✓ Pass
detector.py	Webcam recognition	Correct label displayed	✓ Pass
predict.py	Video recognition	Face labelled in video	✓ Pass
db_config.py	MySQL connection	Connection established	✓ Pass
login_page.py	Valid/invalid auth	Only valid login passes	✓ Pass

5.2 Integration Test Results

Table 4: Integration Test Results

Feature	Steps Tested	Outcome	Status
Login System	Valid & invalid credentials	Invalid rejected, valid accepted	✓ Pass
User Registration	Add user with all fields	Record saved in MySQL	✓ Pass
Face Capture	Webcam capture session	300 images stored correctly	✓ Pass
Model Training	Train from stored images	.xml model file created	✓ Pass
Live Recognition	Webcam recognition	Name shown with green box	✓ Pass

Video Recognition	Predict from .mp4 file	Face correctly labelled	✓ Pass
-------------------	------------------------	-------------------------	--------

5.3 Performance Metrics

Table 5: System Performance Evaluation

Metric	Observed Result
Detection Rate (well-lit, frontal)	~100%
Recognition Accuracy (≥300 samples/user)	93-97%
False Positive Rate	< 5%
False Negative Rate	~5%
Average Recognition Latency	< 1 second
System Crashes / Freezes	None observed

5.4 Observations and Limitations

The system performed best under consistent indoor lighting with the user facing the camera directly. Recognition accuracy improved significantly as training samples increased toward 300 per user, with diminishing returns beyond that threshold. Haar Cascade detection was observed to struggle with non-frontal poses (>30° rotation) and low-light conditions (illuminance <100 lux). The hardcoded admin credentials represent a security limitation for production deployment. The confidence threshold of 50% was empirically determined as the optimal trade-off between false positive and false negative rates.

6. CONCLUSION

This paper presented a complete, GUI-based Face Recognition System using Python and OpenCV, achieving a recognition accuracy of approximately 95% on a standard laptop webcam under controlled indoor conditions. The LBPH algorithm provided an effective balance between accuracy and computational efficiency, confirming the viability of classical machine vision techniques for real-world biometric identity verification without requiring specialised hardware or cloud connectivity.

The six-module architecture (Authentication, Data Acquisition, Classifier Training, Recognition, GUI, Database) proved maintainable and extensible. All six primary objectives — data capture, model training, real-time

recognition, GUI interaction, MySQL integration, and access control — were successfully implemented and validated through unit, integration, and user acceptance tests. The system operated stably across all test sessions with no crashes recorded.

6.1 Future Work

Several enhancements are identified for future iterations: (1) Deep Learning Integration — replace LBPH with FaceNet or ArcFace for higher accuracy under challenging conditions such as varying lighting, pose, and occlusion; (2) Anti-Spoofing — incorporate liveness detection to prevent replay attacks using printed photos or video playback; (3) Multi-Face Recognition — extend detector.py to identify multiple individuals simultaneously in a single frame; (4) Mobile Application — develop Android/iOS versions using Kivy or Flutter for portable deployment; (5) Cloud Integration — migrate model storage and user records to Firebase or AWS for centralised multi-site access; (6) Attendance Export — auto-generate date-stamped CSV/Excel attendance reports; (7) Encrypted Credentials — replace hardcoded admin credentials with bcrypt-hashed database records.

ACKNOWLEDGEMENT

The author sincerely thanks the supervisor and faculty members of the Dept. of CSE, AI & ML, Mangalayatan University, for their expert guidance and continuous support. Gratitude is also extended to colleagues and the institute for providing the necessary resources and encouragement throughout this project.

REFERENCES

- [1] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.
- [2] G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, 2008.
- [3] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.
- [4] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face Description with Local Binary Patterns: Application to Face Recognition," IEEE Trans. Pattern Analysis and Machine Intelligence, 2006.
- [5] OpenCV Documentation: <https://docs.opencv.org/>
- [6] MySQLConnectorforPython: <https://dev.mysql.com/doc/connector-python/en/>
- [7] PythonOfficialDocumentation: <https://docs.python.org/3/>

BIOGRAPHIES

Nishant Kumar is an M.Tech Scholar in Computer Science with specialization in Artificial Intelligence & Machine Learning (AI & ML) at Mangalayatan University, Aligarh, India. His research interests include computer vision, biometric systems, and machine learning. This Face Recognition System using Python and OpenCV was his M.Tech dissertation project (2023–2025).