CHAPTER 1 INTRODUCTION

1.1 Overview of the Project

Surveillance systems are widely used for monitoring and securing environments such as public spaces, institutions, and private premises. Traditional surveillance requires continuous human supervision, which is often inefficient, error-prone, and not scalable for large volumes of video data. To overcome these limitations, this project proposes a smart surveillance system capable of detecting abnormal human activities automatically.

The proposed system uses deep learning techniques to classify frames from surveillance footage into two categories: normal and anomalous. It employs a Convolutional Neural Network (CNN) model trained on the DCSASS dataset, which contains labeled instances of various human activities. Once trained, the model is integrated into a user-friendly interface built using Streamlit, allowing real-time image and video upload, processing, and result display.

Upon detecting an anomaly, the system automatically logs the event and sends an alert email to the registered user with visual evidence. It also includes secure user authentication, an activity log database, and an admin panel for managing users and system records. Designed for local deployment, this system serves as a practical, efficient, and scalable solution to enhance existing surveillance mechanisms.

1.2 Problem Statement

Traditional surveillance systems depend on human operators to observe live video feeds and identify unusual or suspicious activities. This manual observation method is often inefficient, inconsistent, and impractical for extended monitoring or handling multiple camera feeds simultaneously. Human attention can waver over time, increasing the risk of missing critical incidents.

Additionally, most conventional systems do not include intelligent detection features or instant alert mechanisms. As a result, abnormal behaviors such as violence, accidents, or intrusions may remain unnoticed until after the event has occurred. Delayed recognition of such events can compromise safety and security in sensitive environments.

To address these concerns, there is a need for a system that can assist in identifying abnormal human activities in surveillance media accurately and promptly, thereby improving the overall effectiveness of monitoring processes.

1.3 Objectives of the Project

The main objective of this project is to develop a smart surveillance system capable of identifying abnormal human activities in surveillance footage using deep learning techniques. The system is designed to enhance monitoring efficiency, reduce dependency on manual observation, and provide timely awareness of critical situations.

The specific objectives include:

- To train a Convolutional Neural Network (CNN) using the DCSASS dataset for classifying video frames as either normal or anomalous.
- To process surveillance footage frame-by-frame for reliable activity classification.
- To provide a user-friendly web interface for uploading images and videos for analysis.
- To notify registered users when an anomaly is detected with significant confidence.
- To maintain a record of detection events including timestamps, and visual evidence.
- To ensure secure user registration and login through hashed password storage.
- To enable administrators to manage users through admin panel.
- To build a system that runs locally, ensuring data privacy and ease of access without relying on external servers.

1.4 Scope of the Project

This project focuses on building a smart surveillance system capable of analyzing visual data and detecting abnormal human activities such as violence, robbery in real-time image and video inputs. The system is designed to work efficiently in environments such as public spaces, institutions, offices, and residential areas where surveillance is essential.

The project scope includes:

- Developing a deep learning model using Convolutional Neural Networks (CNN) trained on the DCSASS dataset.
- Supporting both image and video uploads through a locally hosted web interface.
- Logging and displaying anomaly detection results with timestamp and confidence scores.
- Sending email alerts to registered users when potential anomalies are detected.
- Maintaining user data and detection logs securely using a local SQLite database.
- Providing administrative access to view logs, manage users, and monitor system activity.

The system is intended for desktop deployment and does not rely on internet connectivity for detection or storage. While mobile deployment and real-time CCTV camera integration are not currently included, the design allows flexibility for such enhancements in the future.

1.5 Methodology / SDLC Model Adopted

The Smart Surveillance System was developed using a linear and phase-based approach similar to the Waterfall Model, which is well-suited for academic and structured software development projects. This methodology ensured that each phase of the system was carefully completed and validated before progressing to the next, thereby reducing the chances of rework and ensuring project stability.

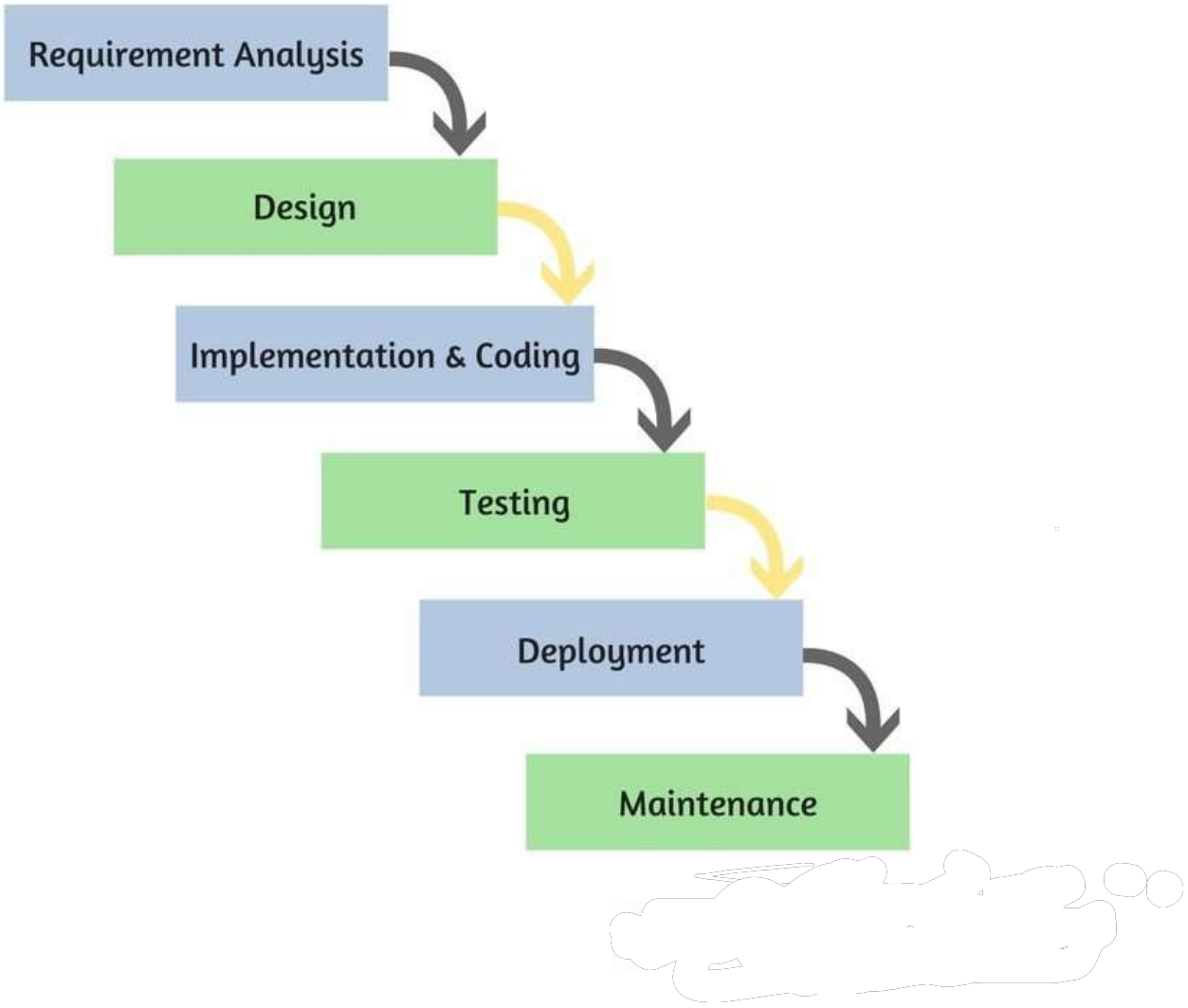

Fig 1.5.1 Waterfall Model

The methodology followed consists of the following phases:

1. Requirement Analysis

The initial phase involved identifying the essential system requirements, including image and video input handling, abnormal activity detection, user authentication, logging mechanisms, and email notifications. These requirements were gathered through observation of common surveillance needs and project feasibility assessment.

2. Dataset Processing and Model Training

The DCSASS dataset was preprocessed by extracting video frames, converting them to grayscale, resizing them to 64×64 pixels, and normalizing pixel values. A Convolutional Neural Network (CNN) was designed and trained on this processed data to classify frames as either normal or anomalous.

3. System Design

Architectural planning was carried out for both backend and frontend components. This included defining the data flow, database structure (SQLite), system architecture, and module-level design for user interaction and anomaly detection functionalities.

4. Implementation

The system was implemented using Python. The backend model was integrated into a Streamlit-based web application. Modules for user registration, login, image/video upload, detection processing, email alerts, and admin controls were developed and tested incrementally.

5. Testing and Evaluation

Various testing methods were applied, including validation during training, performance testing on unseen data, and user-level testing via the interface. Evaluation metrics such as accuracy, F1-score, and ROC AUC were used to assess model performance.

6. Deployment

The final application was deployed locally on a desktop environment, allowing registered users to interact with the system in real time without the need for internet-based services or cloud integration.

This structured methodology ensured a disciplined development process and helped in achieving the project goals within the given timeframe and resource constraints.

1.6 Organization of the Report

This project report is organized into multiple chapters, each detailing a specific phase of the development of the Smart Surveillance System. The structure ensures a clear understanding of the system's objectives, methodology, implementation, and outcomes. The contents of each chapter are summarized below:

- **Chapter 1: Introduction**

  Provides an overview of the project, defines the problem statement, lists the project objectives, explains the scope, describes the methodology adopted, and outlines the structure of the report.

- **Chapter 2: Literature Survey**

  Reviews existing surveillance systems, discusses their limitations, establishes the need for the proposed system, and presents a comparative analysis.

- **Chapter 3: System Analysis**

  Includes a feasibility study covering technical, economic, and operational aspects. It also presents the Software Requirements Specification (SRS) and lists both functional and non-functional requirements.

- **Chapter 4: System Design**

  Describes the architectural design, database structure, data flow, UML diagrams, user interface layouts, and the design standards followed.

- **Chapter 5: Implementation**

  Details the technologies used, the step-by-step implementation of modules, code integration strategy, and sample code snippets demonstrating key logic.

- **Chapter 6: Testing**

  Explains the testing strategy applied at various stages, including unit testing, integration testing, and system testing, along with test results and quality assurance considerations.

- **Chapter 7: Results and Discussion**

Presents screenshots of the system, interprets the results, evaluates performance metrics, and provides a comparative analysis with existing systems.

- **Chapter 8: Conclusion and Future Scope**

Summarizes the work carried out, discusses challenges faced, outlines limitations, and suggests future enhancements.

- **Chapter 9: References**

Lists all sources referred to during the development of the project, including research papers, documentation, websites, and tools used.

CHAPTER 2 LITERATURE SURVEY

2.1 Review of Existing System

Conventional surveillance systems are widely deployed across public and private spaces for safety, security, and monitoring purposes. These systems primarily consist of CCTV cameras that continuously record footage, which is either monitored live by security personnel or stored for later review. While these setups serve as a visual deterrent and provide evidence after incidents, they are limited in their capability to detect ongoing abnormal activities effectively.

In most existing systems, the responsibility of detecting suspicious behavior lies entirely with human operators. This dependence on manual observation can lead to delays in response and oversight due to fatigue or distraction. Moreover, monitoring multiple camera feeds simultaneously becomes impractical in large-scale setups, increasing the chances of missing critical events.

Some advanced surveillance solutions attempt to incorporate basic motion detection or activity recognition through rule-based logic, such as identifying unusual movement patterns or unauthorized access to restricted zones. However, these methods are often inflexible, prone to false alarms, and not capable of distinguishing between different types of human behavior.

With the increasing availability of high-resolution video data and the growing demand for smarter security infrastructure, there is a need for systems that can go beyond simple motion tracking and enable more accurate recognition of specific abnormal human actions.

2.2 Limitations of Existing Approach

Despite the widespread use of traditional surveillance systems, they exhibit several limitations that hinder their effectiveness in detecting abnormal activities in real time.

These limitations include:

- **Lack of Real-Time Decision Making:** Existing systems rely heavily on manual monitoring, which limits their ability to respond instantly to abnormal activities such as fights, thefts, or accidents.
- **Human Dependency:** Continuous observation by security personnel is required. This leads to fatigue, distraction, and potential human error, reducing the overall reliability of detection.
- **Inflexibility and Rule-Based Detection:** Many systems use predefined rules or basic motion sensors that cannot adapt to varying environments or distinguish between normal and abnormal behaviors in complex scenarios.
- **High False Alarm Rate:** Basic systems often generate false positives from non-threatening movements such as shadows, sudden lighting changes, or animal movement, leading to unnecessary alerts.
- **Lack of Contextual Understanding:** Most traditional methods fail to understand the context of actions in a scene and therefore cannot differentiate between similar-looking but contextually different events (e.g., running in a playground vs. running during a theft).
- **No Automatic Notification or Logging Mechanism:** In most systems, abnormal events are only noticed upon manual inspection. There is no automatic alert system or structured event logging for analysis and future reference.

2.3 Need for the Proposed System

Given the drawbacks of traditional surveillance systems, there is a growing need for a more efficient solution that can assist in detecting abnormal human activities with higher accuracy and minimal reliance on manual monitoring. The proposed Smart Surveillance System addresses this gap by introducing a structured, frame-based detection mechanism powered by deep learning.

The need for the proposed system arises from the following key considerations:

- **Improved Monitoring Efficiency:** By analyzing individual frames from video feeds, the system can identify unusual activities without the need for constant human supervision.

- **Reduction in Human Error:** The system reduces dependency on manual observation, helping avoid missed detections due to fatigue or oversight.
- **Consistent Performance:** Unlike humans, the system can maintain a consistent level of vigilance across long periods and multiple video inputs.
- **Instant Feedback and Alerts:** The system sends email notifications to registered users upon detecting high-confidence anomalies, allowing timely awareness and action.
- **Event Logging and Evidence Collection:** All detected anomalies are logged with timestamps, confidence scores, and the corresponding frames, which can be useful for verification or investigation.
- **Ease of Use and Accessibility:** The web-based interface allows users to upload images or videos and get detection results instantly, making it user-friendly and efficient.
- **Privacy and Local Deployment:** The application runs entirely on the local system, ensuring that surveillance data is not transmitted over the internet or exposed to external servers.

2.4 Comparative Study

S. No	Author	Paper Title	Year	Parameters and Objectives of Paper
1	Ganagavalli Karuppasamy, Bannari Amman	YOLO-based anomaly activity detection system for human behavior analysis and crime mitigation	2024	Develop a real-time system for detecting abnormal behavior in video streams.
2	monji mohamed zaidi et al.	Suspicious Human Activity Recognition From Surveillance Videos Using Deep Learning	2024	Introduce Time Distributed CNN and Conv3D models for recognizing six suspicious activities. Evaluate models using custom datasets to achieve high accuracy and robustness.

3	Antonio Carlos Cob-Parro et al.	A new framework for deep learning video based Human Action Recognition on the edge	2024	Propose a lightweight, scalable framework for real-time action recognition, Achieve accurate and efficient human action recognition
4	Yufei Xie	Deep Learning Approaches for Human Action Recognition in Video Data	2024	Evaluate CNNs, RNNs, and Two-Stream ConvNets for human action recognition. Analyze accuracy, precision, recall, and F1-score for action recognition using UCF101 dataset.
5	Anjali Suthar et al.	Abnormal Activity Recognition in Private Places Using Deep Learning	2023	Enhance ATM security by detecting suspicious activities using video analytics.
6	Yanjinlkham Myagmar-Ochir, Wooseong Kim	A Survey of Video Surveillance Systems in Smart City	2023	Discusses the role of video surveillance in smart cities, focusing on safety, traffic management, and healthcare applications. It also highlights the integration of deep learning, blockchain, and cloud computing for real-time monitoring and anomaly detection.
7	Tatsiana Isakava	A gentle introduction to human activity recognition	2022	Reviews deep learning models like CNNs and RNN for HAR, analyzing how these models address complex tasks.

8	Musa Dima Genemo	Suspicious activity recognition for monitoring cheating in exams	2022	Design a deep CNN-based model to detect suspicious behaviors during exams.
9	Houssem Eddine Azzag, Imed Eddine Zeroual, Ammar Ladjailia	Real-Time Human Action Recognition Using Deep Learning	2022	Leverage CNNs with 5 hidden layers for real-time human action recognition, trained on the Weizmann dataset. Achieved 84.44% accuracy and addressed challenges like multi-action clips.
10	Marius Bock, Alexander Hoelzemann, Michael Moeller, Kristof Van Laerhoven	Tutorial on Deep Learning for Human Activity Recognition	2021	Covers the use of deep learning for activity recognition using wearable sensors, with practical insights into setting up data pipelines and validating approaches. It discusses the evolution from feature engineering to end-to-end deep learning models for HAR.

S. No	Paper Title	Algorithms Used	Your Observation and Impact on Your Project
1	YOLO-based anomaly activity detection system for human behavior analysis and crime mitigation	YOLO, LSTM-CNN	Focuses on real 66-time anomaly detection in surveillance.Can guide the development of real-time detection capabilities.
2	Suspicious Human Activity Recognition From Surveillance Videos Using Deep Learning	CNN, Conv3D	Demonstrates the effectiveness of CNN-based approaches for detecting activities in surveillance. It inspires ideas for integrating time-distributed features in your project.
3	A new framework for deep learning video based Human Action Recognition on the edge	MobileNetV2-SSD, LSTM, CNN,RNN	The framework enables real-time, scalable, and cost-efficient human action recognition on edge devices, achieving near-state-of-the-art performance for diverse real-world applications. The focus on embedded hardware will guide optimization strategies in the project.

4	Deep Learning Approaches for Human Action Recognition in Video Data	CNNs, RNNs, Two-Stream ConvNets	The combination of spatial and temporal learning in Two-Stream ConvNets is crucial for your work. The paper's comparison of these models provides a foundation for your HAR model selection.
5	Abnormal Activity Recognition in Private Places Using Deep Learning	CNN, YOLOv5	Highlights the importance of robust activity detection for ATM security. Provides insights into integrating detection systems in constrained environments.
6	A Survey of Video Surveillance Systems in Smart City	Deep Learning, Blockchain, Cloud Computing	The integration of real-time surveillance and AI could be crucial for the smart surveillance project, especially in anomaly detection.
7	A Close Look into Human Activity Recognition Models using Deep Learning	CNN, RNN.	The deep learning models discussed can help improve accuracy in detecting human actions, beneficial for the HAR-based surveillance.

8	Suspicious activity recognition for monitoring cheating in exams	Deep CNN (L4-BranchedActionNet), ACS	Focuses on detecting constrained abnormal behaviors like cheating. Useful for understanding human activity recognition in constrained settings.
9	Real-Time Human Action Recognition Using Deep Learning	CNN	Focuses on human action classification using static camera data. Can aid in improving action classification accuracy and adapting for static surveillance systems.
10	Tutorial on Deep Learning for Human Activity Recognition	CNN	The end-to-end approach using deep learning can be adopted for human activity recognition in your project for accurate behavior classification.

Table 2.4.1 Comparative Study

2.5 Summary

This chapter provided a comprehensive review of existing surveillance systems and highlighted their major limitations, including heavy reliance on manual monitoring, lack of contextual understanding, and high false alarm rates. The inadequacy of traditional and rule-based systems in handling real-time abnormal behavior detection has established the need for a more effective and structured solution.

The proposed Smart Surveillance System addresses these challenges by leveraging a deep learning model trained on labeled surveillance footage to classify human activities as normal or anomalous. With added features such as email notifications, logging, user authentication, and a local web interface, the system offers a more accurate, responsive, and user-accessible solution.

CHAPTER 3 SYSTEM ANALYSIS

3.1 Feasibility Study

A feasibility study was conducted to evaluate the practicality of implementing the Smart Surveillance System. The study focused on four key dimensions to ensure the project's success within available resources and constraints.

1. **Technical Feasibility:** The proposed system is technically feasible using currently available tools and technologies. The project uses Python as the development language, TensorFlow for deep learning model training, and Streamlit for the user interface. Video and image processing are handled using OpenCV, while data storage is managed with SQLite. All tools used are open-source and well-supported, making them suitable for local system deployment. The system runs efficiently on a standard computer without requiring high-end hardware.
2. **Economic Feasibility:** The project is cost-effective as it is developed entirely using free and open-source software libraries. No commercial licenses, subscriptions, or paid APIs are required. Email alerts are implemented using a free Gmail SMTP configuration. The only cost incurred was for system infrastructure, which was already available for development and testing. Therefore, the system meets economic feasibility for both development and deployment.
3. **Operational Feasibility:** The system is operationally feasible and easy to use, requiring minimal technical knowledge from users. It supports image and video input through a web-based interface and provides clear feedback through logs and alerts. User registration and login ensure secure access, while administrators can manage logs and users through a dedicated panel. The interface is intuitive, and the system behaves consistently across different use cases, making it suitable for real-world applications.

4. **Time and Cost Estimation:** The development process was structured into phases such as requirement analysis, dataset processing, model training, system design, integration, testing, and deployment. The project was completed within the estimated academic schedule. No additional monetary expenses were incurred beyond the use of existing resources and free tools, making the project both time-bound and cost-efficient.

3.2 Software Requirements Specification (SRS)

The Software Requirements Specification (SRS) provides a detailed overview of the functional and non-functional requirements, system interfaces, and operating environment for the Smart Surveillance System.

1. Purpose

The purpose of this project is to develop a Smart Surveillance System capable of detecting abnormal human activities in image and video files using a trained deep learning model. The system empowers users to upload media content and receive instant feedback, including real-time alerts via email if suspicious behavior is detected. It also maintains user activity logs for future reference. This project aims to provide an affordable, lightweight, and intelligent alternative to traditional surveillance systems, particularly useful for institutions, homes, and local authorities with limited resources.

2. Scope

This system provides an interactive desktop-based application that performs anomaly detection using pre-trained deep learning models. It includes features for user registration, secure login, file uploads, automatic detection, and alert notification. The admin interface allows authorized administrators to manage user accounts.

This system is not intended for real-time CCTV surveillance or high-traffic enterprise use but serves as a proof-of-concept or localized monitoring solution. It is best suited for academic demonstrations or small-scale deployment in environments like schools, shops, or homes where security footage is collected and analyzed periodically.

3. Product Functions

The main functions of the system include:

- **User Management**: Registration and authentication system with password hashing for security.
- **File Upload Interface**: Users can upload image or video files through a Streamlit UI.
- **Abnormality Detection**: The system analyzes content using a deep learning model trained to distinguish between normal and abnormal activity.
- **Visual Feedback**: For both image and video uploads, the system detects anomalies and highlights the frame with the highest confidence displaying it to the user and sending it via email if the confidence exceeds a defined threshold.
- **Email Notification**: Automatic email alerts are sent to users with an attached frame if an anomaly is detected with high confidence.
- **Activity Logging**: Each anomaly detection is stored along with the user's ID, timestamp, confidence level, and the relevant frame.
- **Admin Dashboard**: Admins can view registered users, monitor their registrations, and delete accounts when needed.

4. User Classes and Characteristics

- **General Users**:
    - Need to create an account with valid credentials.
    - Can upload image/video files.
    - Receive email alerts when anomalies are detected.
    - Can view their activity logs (images with timestamps and confidence).
    - Have basic computer knowledge but no ML expertise.
- **Administrator**:
    - Has access to a separate login panel.
    - Can view all registered users.
    - Can delete suspicious or inactive accounts.
    - Requires knowledge of the system but not the underlying machine learning logic.

5. Operating Environment

- **Operating System**: Windows 10/11
- **Backend Language**: Python 3.8+
- **Frontend**: Streamlit (Web UI framework)
- **Libraries**: OpenCV, TensorFlow, Keras, Numpy, Matplotlib, SQLite3, Pandas, EmailMessage
- **Database**: SQLite for storing user data and activity logs
- **System Requirements**:
    - RAM: 4 GB minimum
    - Disk: 500 MB free space
    - Internet: Required for email alerting
    - Webcam: Optional (for future live extension)

6. Constraints

- The system supports only static file uploads (images/videos) and does not currently process live CCTV streams.
- Requires a stable internet connection to send email alerts.
- The application is not containerized and must be run locally using Python environment setup.
- Videos larger than a few hundred MB may take longer to process.
- Works best on lower-resolution videos for faster frame-by-frame analysis.

7. Assumptions and Dependencies

- Users have access to pre-recorded surveillance footage.
- The email sender credentials (SMTP Gmail account) are configured correctly and not blocked by Google security.
- The anomaly detection model is already trained and stored as best_anomaly_detector.h5.
- Users have basic knowledge of how to interact with desktop applications and upload media files.

- Future extensions may include live CCTV integration and deployment on the web/cloud.

3.3 Functional and Non-Functional Requirements

This section outlines the functional and non-functional requirements that the Smart Surveillance System must fulfill to operate as intended.

3.3.1 Functional Requirements

These define specific behaviors or functions of the system:

- **FR1:** User Registration and Login
  The system must allow users to register with a username, password, and email. It must verify credentials during login and store passwords in a secure (hashed) format.
- **FR2:** Media Upload
  Users must be able to upload image or video files for analysis through the Streamlit interface.
- **FR3:** Anomaly Detection
  The system must analyze each uploaded image or video frame using a trained CNN model and determine whether it shows normal or anomalous activity.
- **FR4:** Email Notification
  When an anomaly is detected with a confidence level above the set threshold, the system must send an alert email to the registered user, including the detected frame.
- **FR5:** Activity Logging
  Detected anomalies must be stored in a local database with details such as username, timestamp, confidence score, and a copy of the captured frame.
- **FR6:** Admin Access
  The admin must be able to view all registered users. The admin should also have the ability to delete users if necessary.

- **FR7:** Log Viewing

  Users should be able to view a summary of their past anomaly logs, including the time and confidence level of detection.

3.3.2 Non-Functional Requirements

These define system attributes such as performance, usability, reliability, and maintainability:

- **NFR1:** Performance

  The system should be able to process uploaded images and short videos within a few seconds and provide results in near real-time.
- **NFR2:** Usability

  The web interface should be user-friendly and intuitive for non-technical users, with clear instructions and labels.
- **NFR3:** Reliability

  The system should operate consistently without crashes during media upload, detection, or notification.
- **NFR4:** Security

  Passwords must be stored using secure hashing. Only authenticated users should be allowed to upload media and view logs.
- **NFR5:** Portability

  The application should be able to run on any standard desktop system with Python installed, across different operating systems (Windows/Linux).
- **NFR6:** Maintainability

  Code should be modular and well-documented to allow future upgrades, bug fixes, or integration of new features.

CHAPTER 4 SYSTEM DESIGN

4.1 System Architecture

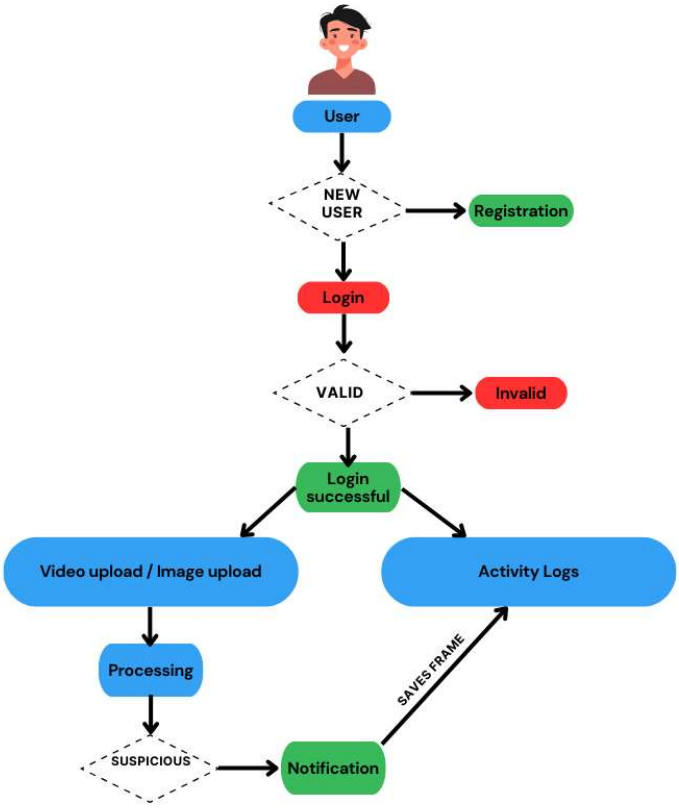

Fig 4.1.1 system architecture

The architecture of the Smart Surveillance System follows a user-driven, event-based workflow that connects user interaction with backend intelligence, real-time anomaly detection, and alert generation. The architecture emphasizes modularity and smooth user experience.

The system architecture consists of the following major components:

- **User Module**: Users initiate interaction by either registering as new users or logging in with existing credentials. Secure login is enforced using SHA-256 password hashing.
- **Authentication Flow**: Upon login, the system validates credentials. Invalid attempts are rejected, while valid users proceed to the core functionality of the application.
- **Media Upload & Processing Module**: Authenticated users can upload images or videos. These inputs are passed to a pre-trained deep learning model that processes frames to detect abnormal behavior.
- **Anomaly Detection Engine**: Each image or video is analyzed frame-by-frame. If an anomaly is detected above a defined confidence threshold, the frame with the highest anomaly confidence is selected.
- **Notification System**: If suspicious activity is found, an email alert with the most suspicious frame is sent to the user.
- **Activity Logging Module**: All detected anomalies are saved with metadata (timestamp, confidence level, and frame) and displayed in the user's Activity Logs tab for future review. This ensures transparency and traceability.

This architecture ensures seamless integration between user interaction, machine learning processing, and notification services, creating a responsive and efficient smart surveillance system.

4.2 Database Design

The Smart Surveillance System uses a lightweight SQLite database to manage user information and anomaly detection logs. The design prioritizes simplicity, data integrity, and performance for a desktop environment where concurrent access is limited.

Users Table

This table stores all registered user credentials and basic details. It ensures secure access and enables personalized features such as activity logging and alert delivery.

- **username** *(Primary Key)*: A unique identifier for each user. This is used across the application to track activities and ensure personalized log access.
- **password**: The password is securely stored using the SHA-256 hashing algorithm to prevent exposure of raw credentials.
- **email**: The user's email address is used to send anomaly alerts.
- **Registration date**: A timestamp that logs the exact date and time when the user registered.

Activity Logs Table

This table records every anomaly detected in both image and video uploads, along with the user details, detection confidence, and the specific frame where the anomaly was detected.

- **id** *(Primary Key)*: Auto-incremented unique identifier for each log entry.
- **username** *(Foreign Key)*: References the username in the users table, establishing a one-to-many relationship (one user can have multiple logs).
- **timestamp**: The exact time when the anomaly was detected and logged.
- **confidence**: A floating-point value representing the model's confidence score for the detected anomaly. Only values above a defined threshold (e.g., 0.6) are logged.
- **frame**: The actual image frame (as a BLOB) where the anomaly occurred. This frame is the highest-confidence frame if processing a video.

Design Highlights

- The database is initialized and managed through Python using the sqlite3 module.
- User data and activity logs are stored in the same database file (users.db), simplifying management.
- All actions such as registration, login, detection, and log retrieval are tied to the username, maintaining a consistent and secure user session.
- Users can only view their own logs, ensuring data privacy.
- Admins can view and delete user accounts, but they do not have access to individual activity logs.

Relationships

- There is a one-to-many relationship between users and activity_logs, meaning one user can have multiple log entries for multiple anomaly detections.

4.3 UML DIAGRAMS

4.3.1 USECASE DIAGRAM

The use case diagram illustrates how users and admins interact with the system. Regular users can register, log in, upload images or videos, and view detection logs. Admins have separate privileges to manage users by logging in to the admin panel and deleting accounts if needed. Each use case is represented with a clear relationship to the respective actor, ensuring that roles and permissions are well defined. This diagram helps identify system boundaries and major functionalities from a user's perspective.

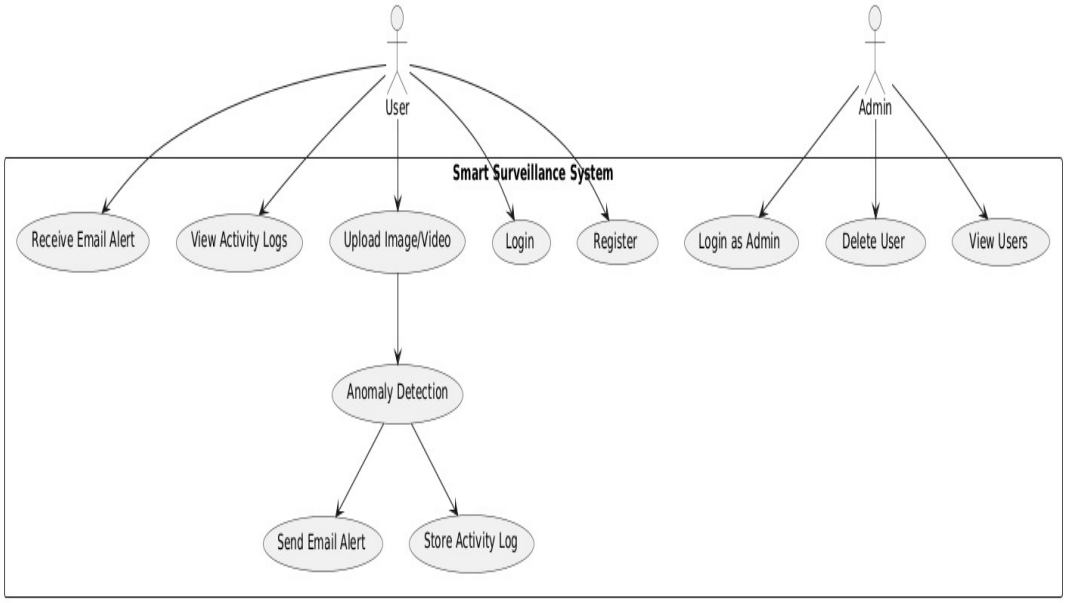

Fig 4.3.1 Use case diagram

4.3.2 CLASS DIAGRAM

The class diagram defines the static structure of the system by showing key classes such as User, Admin, AnomalyDetector, EmailService, and ActivityLog. It highlights attributes like username, email, and confidence, and methods such as login, detect_anomaly, and send_email. Relationships between classes are shown to indicate how they interact—for example, the detector uses the email service and logs data. This structure aids in modular coding and simplifies future enhancements.

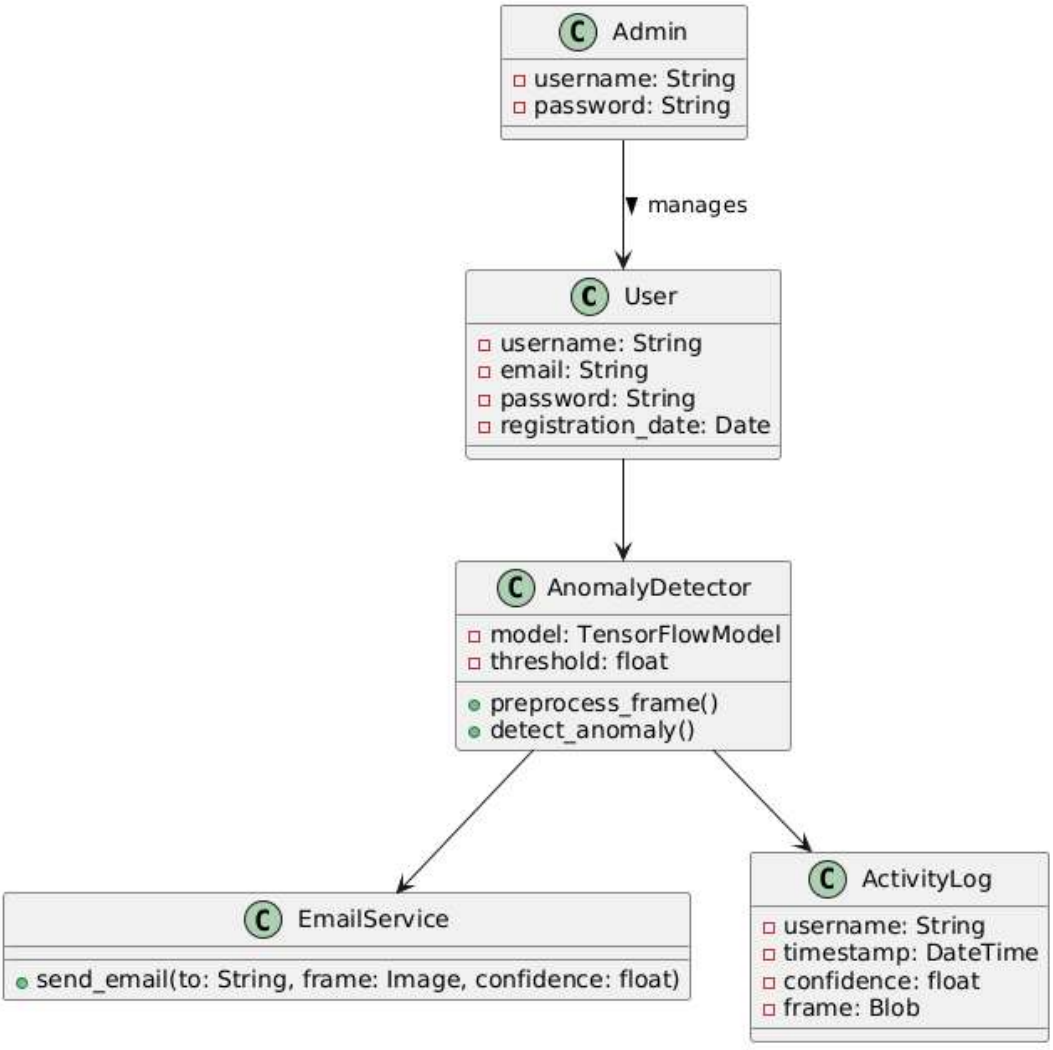

Fig 4.3.2 Class Diagram

4.3.3 ACTIVITY DIAGRAM

The activity diagram shows the logical flow of user activities and decision-making within the system. Starting from user login, it branches based on authentication, leads to media upload, anomaly detection, and either results in logging and alerting or displaying normal status. It represents the different states the system can be in during an operation. This diagram is especially useful for analyzing control flow and ensuring all cases are handled properly.

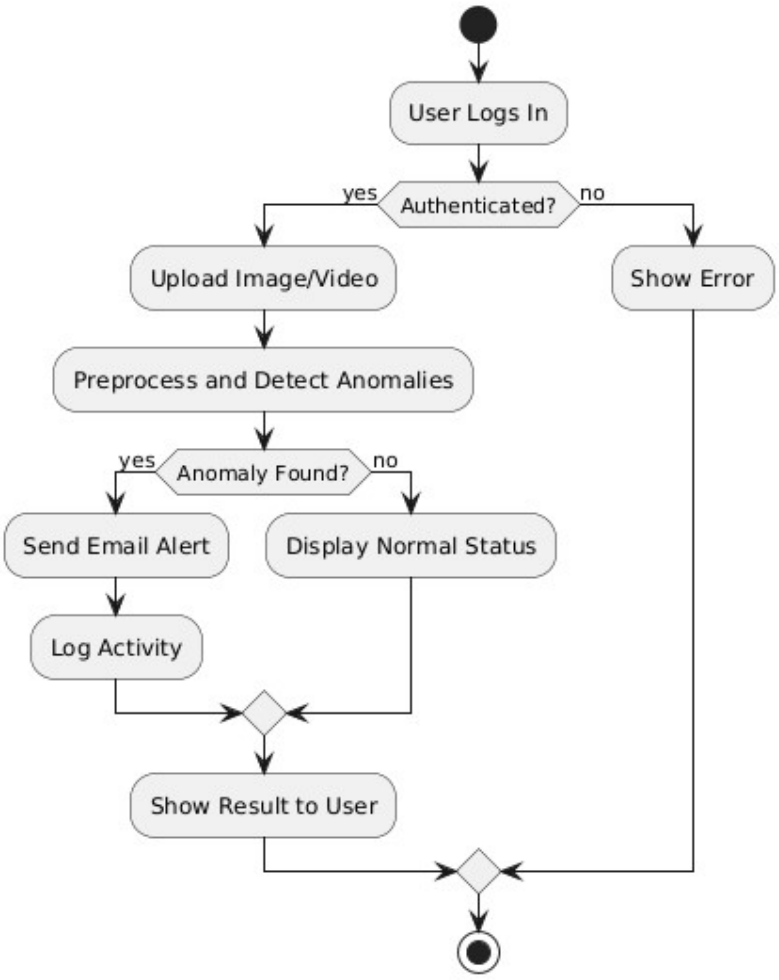

Fig 4.3.3 Activity Diagram

4.3.4 SEQUENCE DIAGRAM

The sequence diagram models the step-by-step flow of actions when a user uploads a video for analysis. It shows the order of interactions between the user interface, anomaly detection module, email service, and database. The diagram clarifies how a request is received, processed, and how a response is returned. If an anomaly is found, an email is sent and a log is saved. This visual helps in understanding the real-time communication and timing of operations.

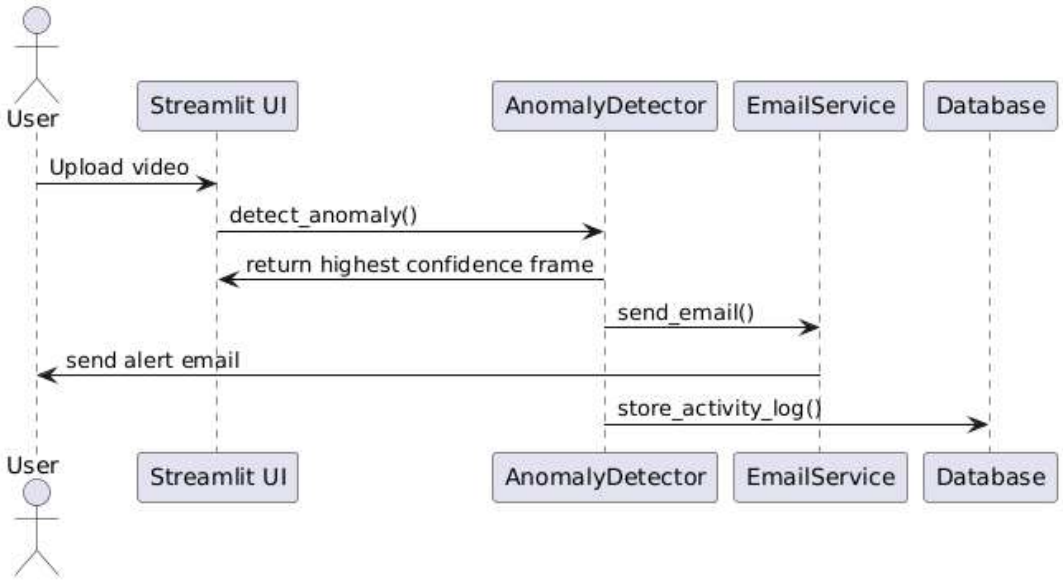

Fig 4.3.4 Sequence Diagram

4.3.5 OBJECT DIAGRAM

The object diagram shows a snapshot of real system instances at a given point in time. It is based on the class structure but represents specific objects with their current attribute values. In this system, it includes a logged-in User object, an active AnomalyDetector object with a set threshold, an EmailService object, and an ActivityLog entry. This diagram demonstrates how various parts of the system are interconnected during runtime, helping visualize object states during an anomaly detection operation.

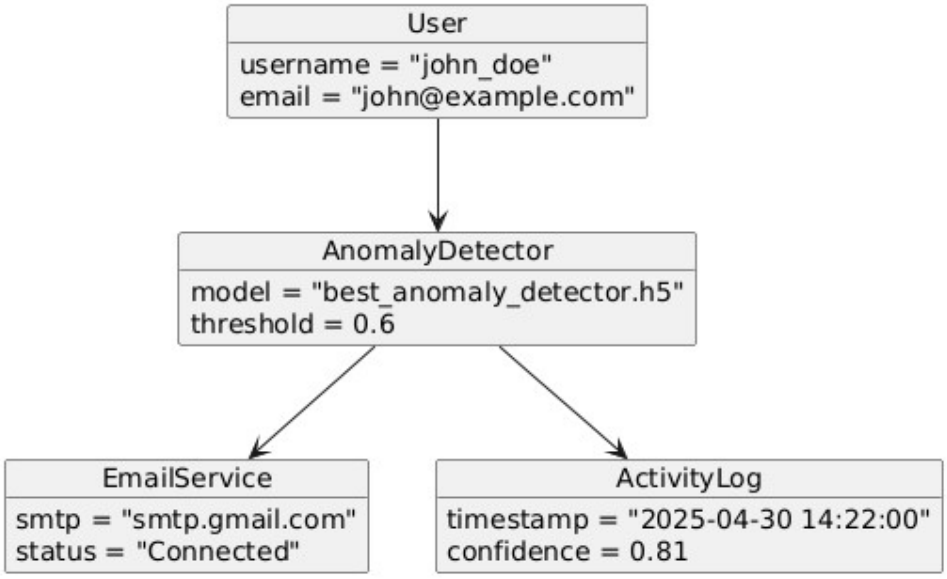

Fig 4.3.5 Object Diagram

4.3.6 DATAFLOW DIAGRAM

The data flow diagram (DFD) outlines how data moves through the system. The flow starts with a user uploading a file. The data is passed to the anomaly detection engine, which processes it and sends results to the database and email module. It also interacts with the user to display feedback. The DFD emphasizes data dependencies and I/O relationships between system modules and external entities like the user and email server.

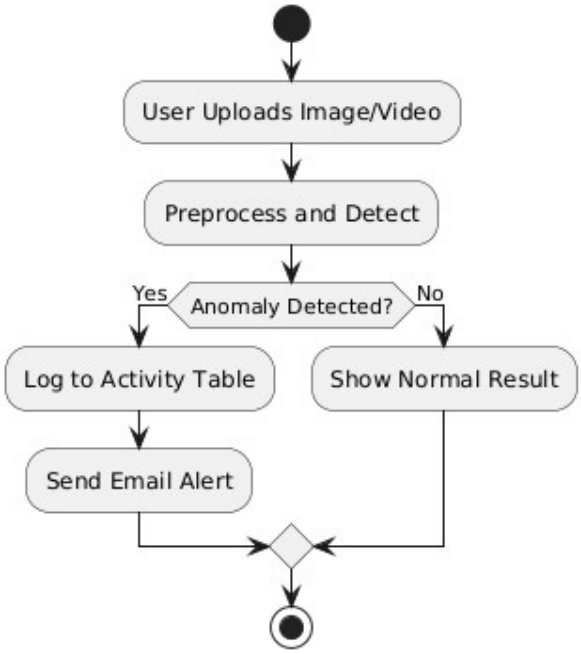

Fig 4.3.6 Dataflow Diagram

4.3.7 STATE DIAGRAM

The state diagram shows the different states the system transitions through during its operation. It starts with the system in the Idle state. When a user logs in, the system transitions to Authenticated, then to Media Upload, and finally Processing. Based on the result, it transitions to either Alert Sent or Normal Display. This diagram helps clarify how system behavior changes in response to user actions and model output.

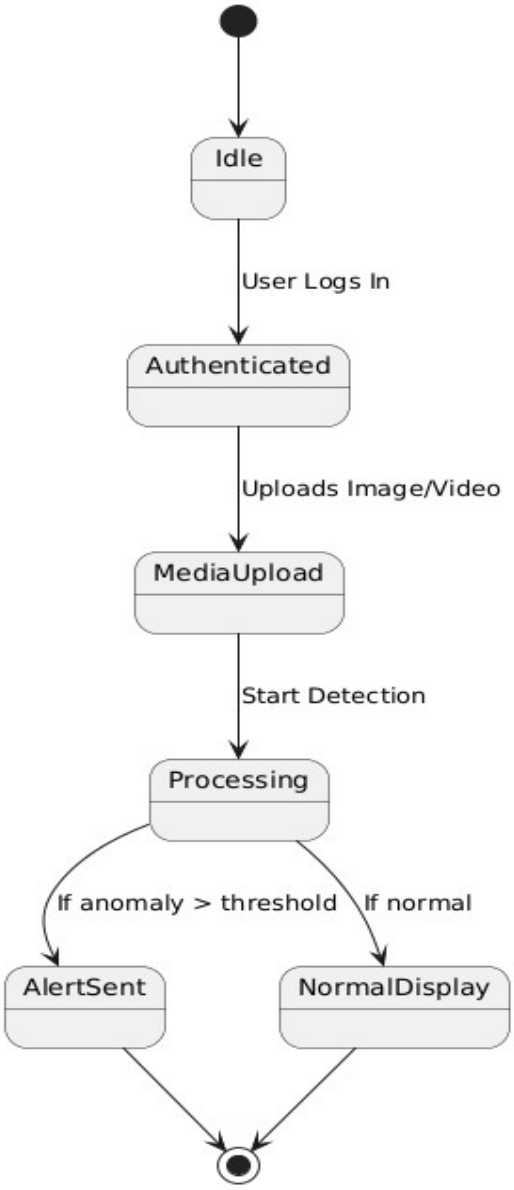

Fig 4.3.7 State Diagram

4.4 User Interface Design

The user interface of the Smart Surveillance System is designed with clarity, responsiveness, and accessibility in mind. Streamlit was chosen as the frontend framework due to its simplicity in integrating with Python backends, making it ideal for real-time monitoring applications.

- **Login & Registration Page**: A minimalistic interface prompts new users to register using a username, email address, and password, while returning users can log in securely. Basic validation and session-based authentication are implemented to ensure data protection.
- **Main Dashboard (Post-Login)**:
    - **Sidebar Navigation**: Provides intuitive access to key features such as image upload, video upload, and logs. A logout button is included to securely terminate sessions.
    - **Image Tab**: Allows users to upload images for anomaly detection. The processed output is displayed alongside detection confidence, prediction labels, and alert messages.
    - **Video Tab**: Users can upload short video clips. The system extracts key frames, performs detection, and summarizes findings in a tabular format including metrics like total frames processed, number of anomalies detected, and detection rates.
    - **Logs Tab**: Displays a chronologically ordered, paginated list of past detections including timestamps, detection types, confidence levels, and thumbnail snapshots for reference.
- **Admin Panel**:
    - Equipped with user management functionality, this interface displays a list of registered users with options to delete accounts if necessary.

All components follow consistent styling, message feedback (success/failure), and structured layout principles to enhance usability.

4.5 Design Standards Followed

The following design principles and standards were adopted to ensure the maintainability, security, and usability of the application:

- **IEEE 1016-2009**: Applied for software design documentation, especially during use case and system modeling.
- **UML 2.x**: Employed for creating standard diagrams including use case, class, sequence, and activity diagrams to visually represent system behavior and data flow.
- **PEP 8 (Python Enhancement Proposal)**: Followed for maintaining consistent Python code formatting and improving code readability.
- **OWASP Top Ten Security Guidelines**: Incorporated for mitigating common web vulnerabilities—especially secure password storage (using hashing) and sanitizing user inputs.
- **ISO/IEC 25010**: Served as a reference framework for ensuring the system meets quality attributes such as functional suitability, reliability, usability, and security.

These standards collectively help in producing a structured, reliable, and scalable surveillance system suitable for academic and practical deployment.

4.6 Safety & Risk Mitigation Measures

A surveillance system must proactively manage risks associated with data processing and user access. The table below outlines key risk factors and their mitigation strategies:

Risk	**Mitigation Strategy**
Missed anomaly detection (False Negatives)	Use of adjustable detection thresholds; optional manual review for borderline confidence cases.
Over-alerting (False Positives)	Apply higher confidence thresholds; enable user review before sending alerts.
Data leakage or privacy concerns	Store files locally, use hashed passwords, and implement access controls.
Excessive email notifications	Batch lower-severity alerts into summary messages; implement cooldown periods.
System overload from large uploads	Restrict file size and type; show appropriate error messages to the user.
Unauthorized access to admin features	Session management and role-based access control to enforce admin-only privileges.

By carefully addressing these factors during design and implementation, the system ensures a secure, efficient, and trustworthy user experience.

CHAPTER 5 IMPLEMENTATION

5.1 Technology Stack

The development of the Smart Surveillance System involved the use of various open-source tools and libraries. The chosen technology stack ensures compatibility, ease of integration, and efficient performance for local deployment.

Programming Language

- **Python**
  Used for implementing the deep learning model, handling video/image processing, backend logic, and web interface.

Deep Learning Framework

- **TensorFlow**
  Utilized for building and training the Convolutional Neural Network (CNN) model used for anomaly detection.

Frontend Interface

- **Streamlit**
  A lightweight, Python-based framework used to build the interactive web interface for user input and displaying detection results.

Media Processing

- **OpenCV (cv2)**: Used for frame extraction, preprocessing, image conversion, and displaying visual feedback on detection results.

Data Handling

- **Pandas & NumPy**: Used for reading label files, handling arrays, and processing datasets during model training and testing.

Database

- **SQLite**: A lightweight relational database used to store user credentials, activity logs, and associated anomaly detection data locally.

Email Notification

- **smtplib and EmailMessage (Python)**: Used to configure the system for sending real-time anomaly alerts via email to registered users.

5.2 Module-wise Implementation

The Smart Surveillance System is composed of several interconnected modules, each performing a specific role. Below is a breakdown of the key modules and their implementation details:

1. Dataset Preprocessing and Model Training Module

- File: train_model.py
- This module loads video frames from the DCSASS dataset, resizes them to 64x64 grayscale images, normalizes pixel values, and labels each frame as normal or anomalous.
- A Convolutional Neural Network (CNN) model is built using TensorFlow and trained on the processed dataset.
- The best-performing model is saved as best_anomaly_detector.h5 based on validation accuracy.

2. Anomaly Detection Module

- Files: test_model.py, test_video.py, test_image.py
- These modules handle testing and validation of the trained model.

- They perform frame-wise prediction and generate metrics such as accuracy, F1-score, confusion matrix, and ROC AUC.
- Video/image files are processed, and detection results are overlaid on frames in real-time.

3. User Authentication Module

- File: app.py
- Allows users to register with a username, email, and password.
- Passwords are securely hashed using the hashlib library before being stored in SQLite.
- Login verification is performed by matching stored hashed passwords.

4. Image and Video Upload Module

- File: smart_survillance.py
- Users can upload an image or video via the Streamlit interface.
- Frames are preprocessed and passed through the CNN model to detect anomalies.
- Detection confidence is displayed, and if an anomaly is found, the system proceeds to log and notify the user.

5. Email Notification Module

- File: smart_survillance.py
- When an anomaly is detected with confidence exceeding the threshold, an email alert is sent to the registered user.
- The email contains a message and the frame attached as evidence.

6. Activity Logging Module

- File: smart_survillance.py
- Anomaly detection logs are saved in an SQLite database with timestamp, user, confidence score, and frame with highest confidence.
- Logged frames are stored in a database for review.

7. Admin Panel Module

- File: app.py
- Admin credentials are predefined.
- Admin can view all registered users, delete user accounts.

5.3 Code Integration Strategy

The development of the Smart Surveillance System followed a modular and staged code integration strategy. Each module was developed and tested independently before being integrated into the complete application. This approach ensured code stability, ease of debugging, and simplified testing during development.

The integration process was carried out as follows:

1. Independent Module Development

Each core functionality—such as model training, testing, user authentication, media processing, and email alerting—was first developed and tested as a standalone script:

- Model training was completed using train_model.py.
- Detection was tested using test_model.py, test_video.py, and test_image.py.
- The user interface and input handling were implemented in smart_survillance.py.

2. Backend and Frontend Separation

To maintain code clarity, backend processing (CNN model, database) was separated from the frontend interaction (user inputs and display), even though both reside in smart_survillance.py. Backend functions like detection, logging, and email alerts were structured as callable functions, which were triggered based on frontend events (uploads, login).

3. Model Integration

Once the model was trained and validated, it was integrated into the Streamlit application using load_model() from TensorFlow. This allowed direct prediction from uploaded frames in real-time without retraining or delay.

4. Database Binding

SQLite integration was done using Python's sqlite3 library. Tables for user credentials and activity logs were created at runtime. Logging functions were abstracted for reuse across modules to ensure consistency.

6. Final Testing and Bug Fixing

After all components were integrated, the complete system was tested as a whole. Edge cases such as invalid files, login failures, or email delivery issues were handled gracefully with proper error messages.

This systematic integration strategy ensured that the system functioned smoothly, and all modules worked together reliably within the final application.

5.4 Sample Code Snippets

Below are representative excerpts from key modules, illustrating the core logic and integration in the Smart Surveillance System.

train_model.py

```
import os
import cv2
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
import warnings
warnings.filterwarnings('ignore')

class AnomalyDetector:
    def __init__(self, dataset_path):
        self.dataset_path = dataset_path
        self.model = None

    def preprocess_frame(self, frame):
        # Resize frame to 64x64
        frame = cv2.resize(frame, (64, 64))
        # Convert to grayscale
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Normalize pixel values
        frame = frame.astype('float32') / 255.0
        # Add channel dimension
        frame = np.expand_dims(frame, axis=-1)
        return frame

    def load_data(self):
        X = []
        y = []
        frame_count = 0
        skipped_frames = 0

        # Load labels
        labels_path = os.path.join(self.dataset_path, 'Labels')

        print("\nProcessing categories:")
```

```
for category in os.listdir(self.dataset_path):
    category_path = os.path.join(self.dataset_path, category)
    if not os.path.isdir(category_path) or category == 'Labels':
        continue

    print(f"\nProcessing {category}...")

    # Load category labels
    label_file = os.path.join(labels_path, f"{category}.csv")
    if not os.path.exists(label_file):
        print(f"Warning: Label file not found for {category}")
        continue

    df = pd.read_csv(label_file, header=None)

    for _, row in df.iterrows():
        frame_count += 1

        # Get video name and frame number
        filename = row[0]
        print(f"\nProcessing file: {filename}")

        try:
            parts = filename.split('_')
            print(f"Parts after split: {parts}")
            video_name = '_'.join(parts[:-1])
            print(f"Video name: {video_name}")
            frame_num = int(parts[-1])
            print(f"Frame number: {frame_num}")
        except Exception as e:
            print(f"Error extracting frame number from {filename}: {str(e)}")
            skipped_frames += 1
```

```
            continue

        # Fix typo in video name if it's a RoadAccidents video
        if category == 'RoadAccidents':
            video_name = video_name.replace('oadAccidents', 'RoadAccidents')
            video_name = video_name.replace('RRoadAccidents', 'RoadAccidents')

        # Get video directory path
        video_dir = os.path.join(category_path, f"{video_name}.mp4")

        if not os.path.exists(video_dir):
            print(f"Warning: Video directory not found: {video_dir}")
            skipped_frames += 1
            continue

        try:
            # Get the specific frame file
            frame_file = os.path.join(video_dir, f"{video_name}_{frame_num}.mp4")

            if not os.path.exists(frame_file):
                print(f"Warning: Frame file not found: {frame_file}")
                skipped_frames += 1
                continue

            cap = cv2.VideoCapture(frame_file)
            ret, frame = cap.read()

            if ret:
                processed_frame = self.preprocess_frame(frame)
                X.append(processed_frame)

                # Validate and convert label to integer
```

```
                label = row[2]
                if pd.isna(label):
                    print(f"Warning: NaN label found for file: {filename}")
                    skipped_frames += 1
                    continue

                label = int(label)
                if label not in [0, 1]:
                    print(f"Warning: Invalid label {label} for file: {filename}")
                    skipped_frames += 1
                    continue

                y.append(label)
            else:
                print(f"Warning: Could not read frame {frame_num} from {video_dir}")
                skipped_frames += 1

            cap.release()
        except Exception as e:
            print(f"Error processing frame {frame_num} from {video_dir}: {str(e)}")
            skipped_frames += 1
            continue

X = np.array(X)
y = np.array(y)

# Validate that X and y have the same length
if len(X) != len(y):
    print(f"Warning: X and y have different lengths: {len(X)} vs {len(y)}")
    print("Attempting to fix by truncating longer array...")
    min_length = min(len(X), len(y))
    X = X[:min_length]
```

```
        y = y[:min_length]

    print(f"\nData loading complete!")
    print(f"Total frames processed: {frame_count}")
    print(f"Frames skipped: {skipped_frames}")
    print(f"Final dataset size: {len(X)} frames")
    print(f"Class distribution: Normal={np.sum(y==0)}, Anomaly={np.sum(y==1)}")

    return X, y

def build_model(self):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

```
    def train(self):
        # Load and preprocess data
        X, y = self.load_data()

        # Split data into train and validation sets
        X_train, X_val, y_train, y_val = train_test_split(
            X, y, test_size=0.2, random_state=42, stratify=y
        )

        # Compute class weights
        class_weights = compute_class_weight('balanced', classes=np.unique(y_train),
y=y_train)
        class_weights = dict(enumerate(class_weights))

        # Build model if not already built
        if self.model is None:
            self.model = self.build_model()

        from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

        # Create callbacks
        early_stopping = EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True,
            verbose=1
        )

        checkpoint = ModelCheckpoint(
            'best_anomaly_detector.h5',
            monitor='val_accuracy',
```

```
            save_best_only=True,
            verbose=1
        )

        # Train the model
        history = self.model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=20,
            batch_size=32,
            class_weight=class_weights,
            callbacks=[early_stopping, checkpoint],
            verbose=1
        )

        # Save the final model
        self.model.save('anomaly_detector.h5')

        return history

if __name__ == "__main__":
    # Initialize detector
    detector = AnomalyDetector('DCSASS Dataset')
    # Train the model
    history = detector.train()
    # Print training results
    print("\nTraining complete!")
    print(f"Final validation accuracy: {history.history['val_accuracy'][-1]:.4f}")
    print(f"Final validation loss: {history.history['val_loss'][-1]:.4f}")
```

test_model.py

```
import os
import cv2
import numpy as np
import pandas as pd
from tensorflow.keras.models import load_model
from sklearn.metrics import classification_report, confusion_matrix, f1_score,
roc_curve, auc, accuracy_score
import matplotlib.pyplot as plt

class AnomalyDetector:
    def __init__(self, model_path='anomaly_detector.h5', image_size=(64, 64)):
        self.model = load_model(model_path)
        self.image_size = image_size
        self.threshold = 0.5  # Confidence threshold for anomaly detection

    def preprocess_frame(self, frame):
        frame = cv2.resize(frame, self.image_size)
        frame = frame / 255.0
        return np.expand_dims(frame, axis=0)

    def detect_anomaly(self, frame):
        processed_frame = self.preprocess_frame(frame)
        prediction = self.model.predict(processed_frame)[0][0]

        if prediction > self.threshold:
            return True, prediction
        return False, prediction

    def process_video(self, video_path):
        # Normalize the path
        video_path = os.path.normpath(video_path)
```

```
        if not os.path.exists(video_path):
            print(f"Error: Video file not found: {video_path}")
            return

        cap = cv2.VideoCapture(video_path)

        if not cap.isOpened():
            print("Error: Could not open video.")
            return

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Detect anomaly
            is_anomaly, confidence = self.detect_anomaly(frame)

            # Display results
            if is_anomaly:
                cv2.putText(frame, f"ANOMALY DETECTED! (Confidence:
{confidence:.2f})",
                            (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            else:
                cv2.putText(frame, "NORMAL", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            cv2.imshow('Surveillance', frame)

            # Press 'q' to quit
            if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
                break

        cap.release()
        cv2.destroyAllWindows()

class AnomalyDetectorTester:
    def __init__(self, model_path):
        self.model = load_model(model_path)

    def preprocess_frame(self, frame):
        # Resize frame to 64x64
        frame = cv2.resize(frame, (64, 64))
        # Convert to grayscale
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Normalize pixel values
        frame = frame.astype('float32') / 255.0
        # Add channel dimension
        frame = np.expand_dims(frame, axis=-1)
        # Add batch dimension
        frame = np.expand_dims(frame, axis=0)
        return frame

    def load_data(self, dataset_path):
        X = []
        y = []
        frame_count = 0
        skipped_frames = 0

        # Load labels
        labels_path = os.path.join(dataset_path, 'Labels')

        print("\nProcessing categories:")
```

```
for category in os.listdir(dataset_path):
    category_path = os.path.join(dataset_path, category)
    if not os.path.isdir(category_path) or category == 'Labels':
        continue

    print(f"\nProcessing {category}...")

    # Load category labels
    label_file = os.path.join(labels_path, f"{category}.csv")
    if not os.path.exists(label_file):
        print(f"Warning: Label file not found for {category}")
        continue

    df = pd.read_csv(label_file, header=None)

    for _, row in df.iterrows():
        frame_count += 1

        # Get video name and frame number
        filename = row[0]
        print(f"\nProcessing file: {filename}")

        try:
            parts = filename.split('_')
            print(f"Parts after split: {parts}")
            video_name = '_'.join(parts[:-1])
            print(f"Video name: {video_name}")
            frame_num = int(parts[-1])
            print(f"Frame number: {frame_num}")
        except Exception as e:
            print(f"Error extracting frame number from {filename}: {str(e)}")
            skipped_frames += 1
```

```
            continue

        # Fix typo in video name if it's a RoadAccidents video
        if category == 'RoadAccidents':
            video_name = video_name.replace('oadAccidents', 'RoadAccidents')
            video_name = video_name.replace('RRoadAccidents', 'RoadAccidents')

        # Get video directory path
        video_dir = os.path.join(category_path, f"{video_name}.mp4")

        if not os.path.exists(video_dir):
            print(f"Warning: Video directory not found: {video_dir}")
            skipped_frames += 1
            continue

        try:
            # Get the specific frame file
            frame_file = os.path.join(video_dir, f"{video_name}_{frame_num}.mp4")

            if not os.path.exists(frame_file):
                print(f"Warning: Frame file not found: {frame_file}")
                skipped_frames += 1
                continue

            cap = cv2.VideoCapture(frame_file)
            ret, frame = cap.read()

            if ret:
                processed_frame = self.preprocess_frame(frame)
                X.append(processed_frame[0])  # Remove batch dimension

                # Validate and convert label to integer
```

```
                label = row[2]
                if pd.isna(label):
                    print(f"Warning: NaN label found for file: {filename}")
                    skipped_frames += 1
                    continue

                label = int(label)
                if label not in [0, 1]:
                    print(f"Warning: Invalid label {label} for file: {filename}")
                    skipped_frames += 1
                    continue

                y.append(label)
            else:
                print(f"Warning: Could not read frame {frame_num} from {video_dir}")
                skipped_frames += 1

            cap.release()
        except Exception as e:
            print(f"Error processing frame {frame_num} from {video_dir}: {str(e)}")
            skipped_frames += 1
            continue

X = np.array(X)
y = np.array(y)

# Validate that X and y have the same length
if len(X) != len(y):
    print(f"Warning: X and y have different lengths: {len(X)} vs {len(y)}")
    print("Attempting to fix by truncating longer array...")
    min_length = min(len(X), len(y))
    X = X[:min_length]
```

```
        y = y[:min_length]

    print(f"\nData loading complete!")
    print(f"Total frames processed: {frame_count}")
    print(f"Frames skipped: {skipped_frames}")
    print(f"Final dataset size: {len(X)} frames")
    print(f"Class distribution: Normal={np.sum(y==0)}, Anomaly={np.sum(y==1)}")

    return X, y

def test(self, dataset_path):
    """
    Test the model on a dataset
    """
    # Load test data
    X_test, y_test = self.load_data(dataset_path)

    if len(X_test) == 0:
        print("Error: No test data found")
        return

    print(f"\nTesting on {len(X_test)} samples...")

    # Evaluate the model
    y_pred_proba = self.model.predict(X_test)
    y_pred = (y_pred_proba > 0.5).astype(int)

    # Calculate metrics
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print("\nClassification Report:")
```

```
    print(classification_report(y_test, y_pred))

    # Calculate F1 score
    f1 = f1_score(y_test, y_pred)
    print(f"\nF1 Score: {f1:.4f}")

    # Calculate accuracy using sklearn's accuracy_score
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\nTest accuracy: {accuracy:.4f}")

    # Plot ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.savefig('test_roc_curve.png')
    plt.close()

    return {
        'accuracy': accuracy,
        'f1_score': f1,
        'roc_auc': roc_auc
    }
```

```
if __name__ == "__main__":
    # Initialize tester with the saved model
    tester = AnomalyDetectorTester('best_anomaly_detector.h5')

    # Test the model
    results = tester.test('DCSASS Dataset')

    print("\nTest Results:")
    print(f"Accuracy: {results['accuracy']:.4f}")
    print(f"F1 Score: {results['f1_score']:.4f}")
    print(f"ROC AUC: {results['roc_auc']:.4f}")
```

smart_survillance.py

```
def send_email(to_email, frame, confidence):
    msg = EmailMessage()
    msg['Subject'] = "Anomaly Detected Alert"
    msg['From'] = FROM_EMAIL
    msg['To'] = to_email
    msg.set_content(f"An anomaly was detected with confidence: {confidence:.2f}")

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    _, img_encoded = cv2.imencode('.jpg', frame_rgb)
    msg.add_attachment(img_encoded.tobytes(), maintype='image', subtype='jpeg',
filename='anomaly.jpg')

    with smtplib.SMTP_SSL('smtp.gmail.com', 465) as smtp:
        smtp.login(FROM_EMAIL, EMAIL_PASSWORD)
        smtp.send_message(msg)

def process_video(video_file, detector):
    """
```

```
Process an uploaded video and display results
"""
# Save video to temporary file
tfile = tempfile.NamedTemporaryFile(delete=False)
tfile.write(video_file.read())

cap = cv2.VideoCapture(tfile.name)

if not cap.isOpened():
    st.error("Error: Could not open video.")
    return None, None, None, None

frame_count = 0
anomaly_count = 0
any_anomaly_detected = False
highest_confidence = 0
best_frame = None  # Store the frame with the highest confidence

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1

    # Detect anomaly
    is_anomaly, confidence = detector.detect_anomaly(frame)

    if is_anomaly:
        anomaly_count += 1
        any_anomaly_detected = True
        if confidence > highest_confidence:
```

```
            highest_confidence = confidence
            best_frame = frame.copy()  # Save the frame with the highest confidence
        else:
            any_anomaly_detected = False

cap.release()

# Calculate statistics
anomaly_rate = (anomaly_count / frame_count) * 100 if frame_count > 0 else 0

# Create statistics dataframe
stats = pd.DataFrame({
    'Metric': ['Total Frames', 'Anomalies Detected', 'Anomaly Rate'],
    'Value': [frame_count, anomaly_count, f'{anomaly_rate:.2f}%']
})

# Determine final status based on any anomaly detection
if any_anomaly_detected:
    final_status = "ANOMALY DETECTED!"
    final_confidence = highest_confidence
else:
    final_status = "NORMAL"
    final_confidence = 0

# Convert the best frame to RGB for display
if best_frame is not None:
    best_frame_rgb = cv2.cvtColor(best_frame, cv2.COLOR_BGR2RGB)
else:
    best_frame_rgb = None

return stats, final_status, final_confidence, best_frame_rgb, anomaly_rate
```

```
def log_activity(username, confidence, frame):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Make sure frame is in BGR format before saving
    frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Encode the frame as a binary blob
    _, img_encoded = cv2.imencode('.jpg', frame_bgr)
    frame_blob = img_encoded.tobytes()

    c.execute('''INSERT INTO activity_logs (username, timestamp, confidence, frame)
                 VALUES (?, ?, ?, ?)''', (username, timestamp, float(confidence), frame_blob))
# Ensure confidence is stored as float
    conn.commit()
    conn.close()

    return frame  # Return the frame for display


def log_activity(username, confidence, frame):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Make sure frame is in BGR format before saving
    frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Encode the frame as a binary blob
    _, img_encoded = cv2.imencode('.jpg', frame_bgr)
    frame_blob = img_encoded.tobytes()
```

```
    c.execute('''INSERT INTO activity_logs (username, timestamp, confidence, frame)
                 VALUES (?, ?, ?, ?)''', (username, timestamp, float(confidence), frame_blob))
# Ensure confidence is stored as float
    conn.commit()
    conn.close()

    return frame  # Return the frame for display
```

5.5 Coding Standards Followed

- **Meaningful Naming Conventions:** Variables, functions, and classes use descriptive names in snake case for functions and variables, and Pascal Case for class names, facilitating intuitive understanding of their purpose.
- **Modular Structure and Single Responsibility**: Code is organized into self-contained functions and classes, each handling one specific task (e.g., frame preprocessing, model prediction, email dispatch), which simplifies testing and future enhancements.
- **Inline Comments**:Public functions and classes include Comments describing their purpose, inputs, and outputs. Complex code blocks are annotated with brief inline comments to explain non-obvious logic.
- **DRY Principle (Don't Repeat Yourself)**: utility functions (e.g., preprocess_frame, log_activity, send_email) are defined once and invoked wherever needed, avoiding code duplication.
- **Version Control and Documentation:** The project code is maintained in a Git repository with clear commit messages. A requirements.txt file specifies exact library versions for reproducibility. Inline documentation and this report ensure transparency of design decisions.

CHAPTER 6 TESTING

6.1 Testing Strategy

The testing strategy for the Smart Surveillance System focused on ensuring the robustness, accuracy, and reliability of the system. Given the system's complexity, a layered testing approach was used:

- **Unit Testing**: Each module was tested independently to ensure its functionality. This included verifying the accuracy of the deep learning model, the correctness of image preprocessing, user authentication, and anomaly detection.
- **Integration Testing**: After unit testing, the individual modules were integrated, and tests were run to ensure proper communication between them. This involved verifying the flow of data from media upload to detection and logging.
- **System Testing:** The entire system was tested in a real-world scenario, simulating the full user experience of uploading videos, detecting anomalies, and receiving email alerts.
- **Performance Testing**: Load testing was done to ensure the system could handle multiple uploads and continuous media processing without degradation in performance.
- **Security Testing**: Security tests were performed to ensure that user authentication, password storage, and email notification mechanisms were robust and secure.

6.2 Unit Testing

Unit testing was conducted on individual functions and classes to ensure that each component performs as expected. The main areas tested were:

- **Model Training**: Testing the function that trains the CNN model to verify the correctness of data processing, model fitting, and accuracy metrics.
- **Image Preprocessing**: Validated functions such as preprocess_frame to ensure that frames are resized, converted to grayscale, and normalized correctly.

- **Anomaly Detection**: Verified that the model prediction function accurately classifies frames as normal or anomalous and provides the expected confidence score.
- **Email Notification**: The email sending function was tested with mock data to ensure that it sends alerts when the confidence threshold is met and includes the correct content.
- **Logging**: Tested the logging functionality to ensure that detected anomalies were recorded properly in the database with accurate timestamps and file paths.
  Each unit test was executed with predefined input data, and expected outputs were compared to the results to confirm functionality.

6.3 Integration Testing

Integration testing was focused on validating the interaction between different system modules, ensuring data flows seamlessly from one module to another:

- **Model and User Interface**: Ensured that after a user uploads a file, the system processes it correctly, feeds it through the model, and displays the result in the UI.
- **Model and Logging**: Verified that when an anomaly is detected, the system logs the event in the database, including the confidence score and corresponding image.
- **Model and Email Notifications**: Ensured that an email notification is triggered when an anomaly is detected above a certain confidence threshold.
- **Frontend and Backend**: The front-end Stream lit interface was integrated with the backend logic to ensure smooth interaction between user inputs and system outputs.

6.4 System Testing

System testing was carried out to evaluate the complete working of the Smart Surveillance System in a controlled environment:

- **End-to-End Functionality**:The full flow from user registration, media upload, anomaly detection, email notification, and logging was tested to ensure that the system functions as expected.
- **Real-World Simulations**:A variety of real-world scenarios were tested, such as uploading different types of video files, handling multiple concurrent uploads, and testing the system's response to various types of abnormal activities.
- **User Acceptance Testing (UAT)**:A small group of end-users tested the system to ensure that it meets their expectations in terms of usability, speed, and reliability.

6.5 Test Cases and Results

Below is a sample of test cases executed during the system testing phase, along with their expected and actual results:

Test Case	**Description**	**Expected Result**	**Actual Result**	**Status**
Test Case 1: Upload Image for Detection	Upload a single image for anomaly detection	Image uploaded successfully, prediction displayed	Image uploaded, prediction displayed	Pass
Test Case 2: Upload Video for Detection	Upload a video file for anomaly detection	Video processed frame-by-frame, anomalies detected	Video processed, anomalies detected as expected	Pass
Test Case 3: Login with Valid Credentials	User logs in with valid credentials	User logged in successfully	User logged in successfully	Pass

Test Case 4: Login with Invalid Credentials	User attempts to log in with incorrect credentials	Login fails, error message shown	Login failed, error message shown	Pass
Test Case 5: Email Alert on Anomaly Detected	Test if email alert is sent when anomaly detected above threshold	Email sent with detection details	Email sent with detection details	Pass
Test Case 6: Activity Log Entry	After detection, check if anomaly is logged in the database	Log entry with timestamp and image path created	Log entry created with timestamp and path	Pass

6.6 Bug Reporting and Tracking

All bugs encountered during the testing phase were reported using **JIRA**. Each bug was categorized based on severity (Critical, Major, Minor), and progress was tracked using JIRA's issue tracking system. Here's an example of how bugs were reported:

- **Bug 1: Video Upload Failure**
  Severity: Major
  Description: The system failed to upload large video files.
  Status: Resolved after implementing file size checks.

- **Bug 2: False Email Alert Trigger**
  Severity: Minor
  Description: The system sent an email alert for low-confidence anomalies.
  Status: Fixed by adjusting the confidence threshold.

6.7 Quality Assurance Standards

The following quality assurance practices were followed to ensure the reliability and maintainability of the system:

- **Code Review**: Regular code reviews were conducted to ensure adherence to coding standards, bug-free logic, and efficient implementations.
- **Documentation**: All modules were thoroughly documented, and inline comments were added for better understanding and maintainability. A detailed user manual was also created for end-users.
- **Performance Benchmarking**: Performance tests were run periodically to ensure that the system's speed and response times met the required thresholds for real-time processing.

CHAPTER 7 RESULTS AND DISCUSSION

7.1 Output Screenshots

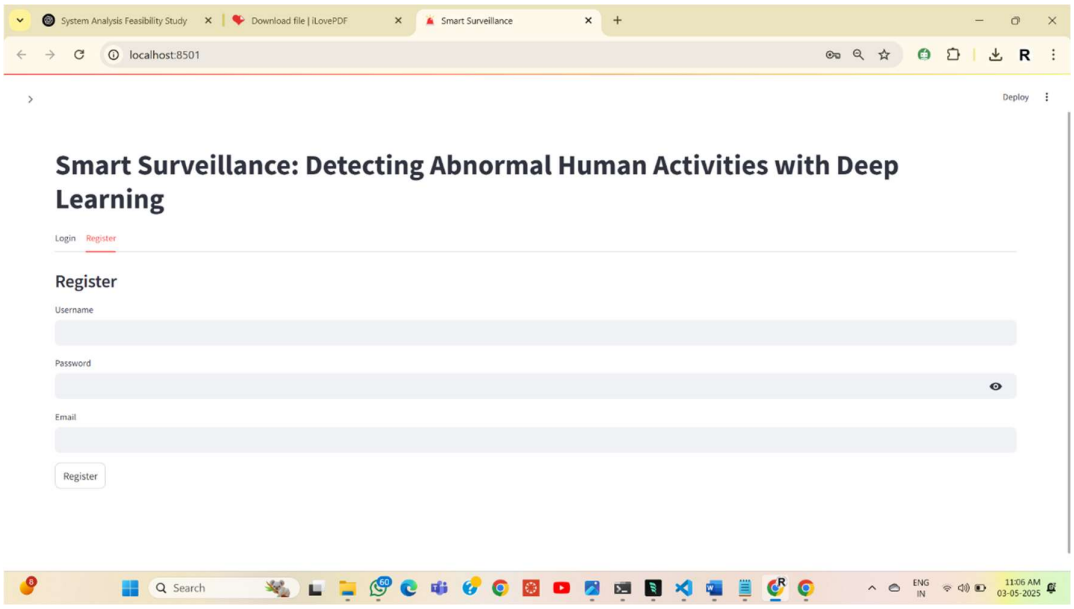

Fig 7.1.1 User Registration

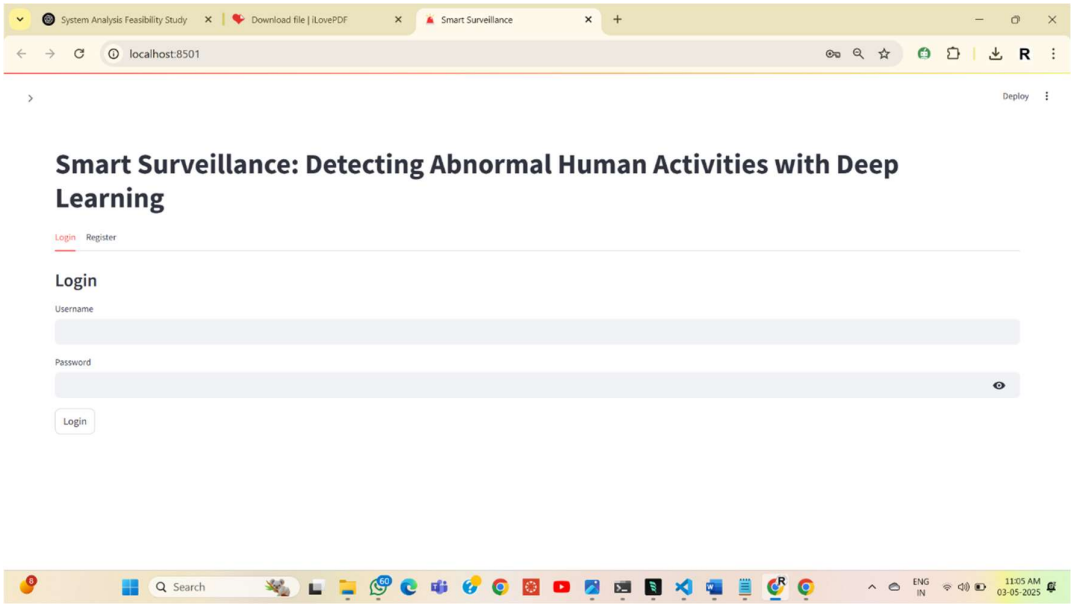

Fig 7.1.2 User Login

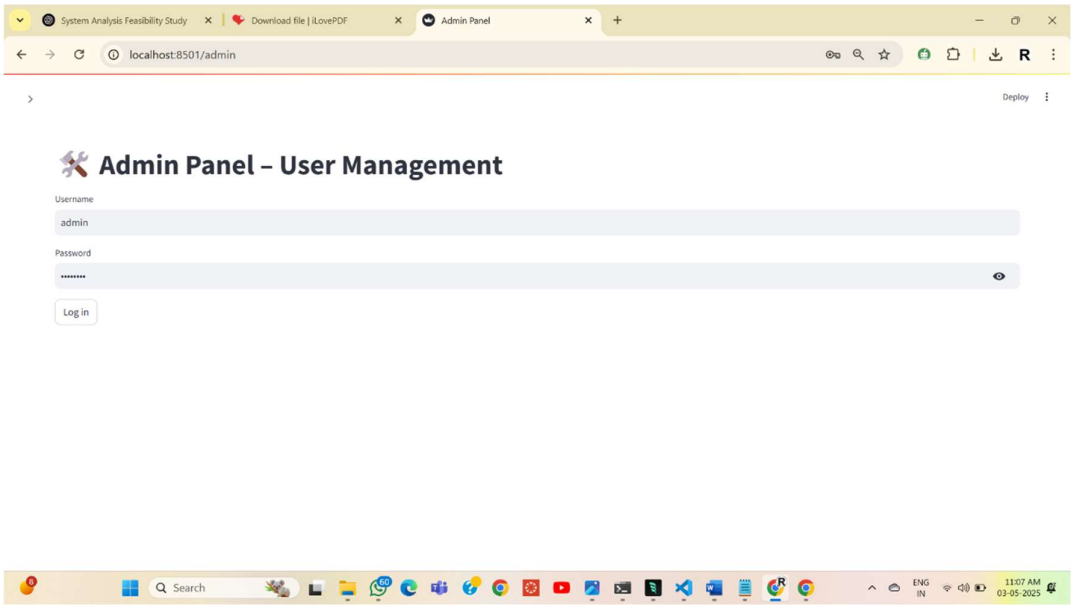

Fig 7.1.3 Admin Login

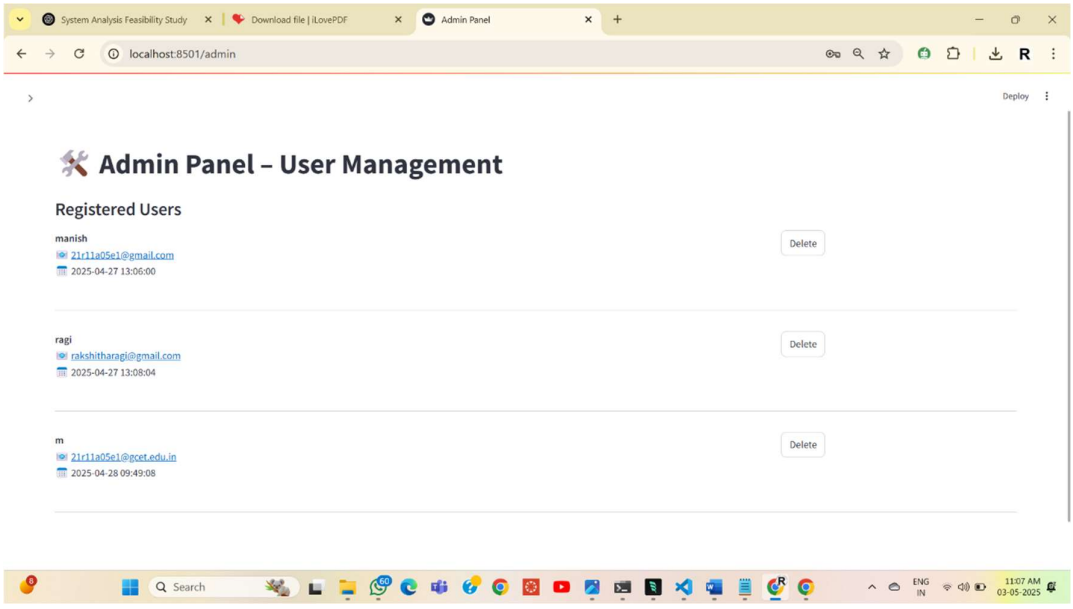

Fig 7.1.4 Admin Dashboard

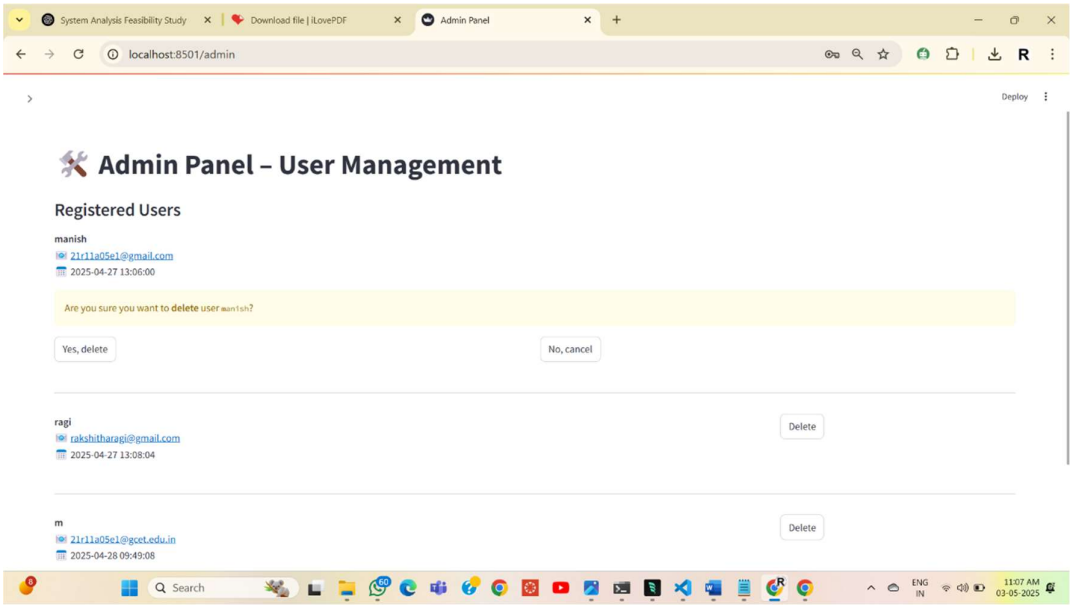

Fig 7.1.5 Delete User

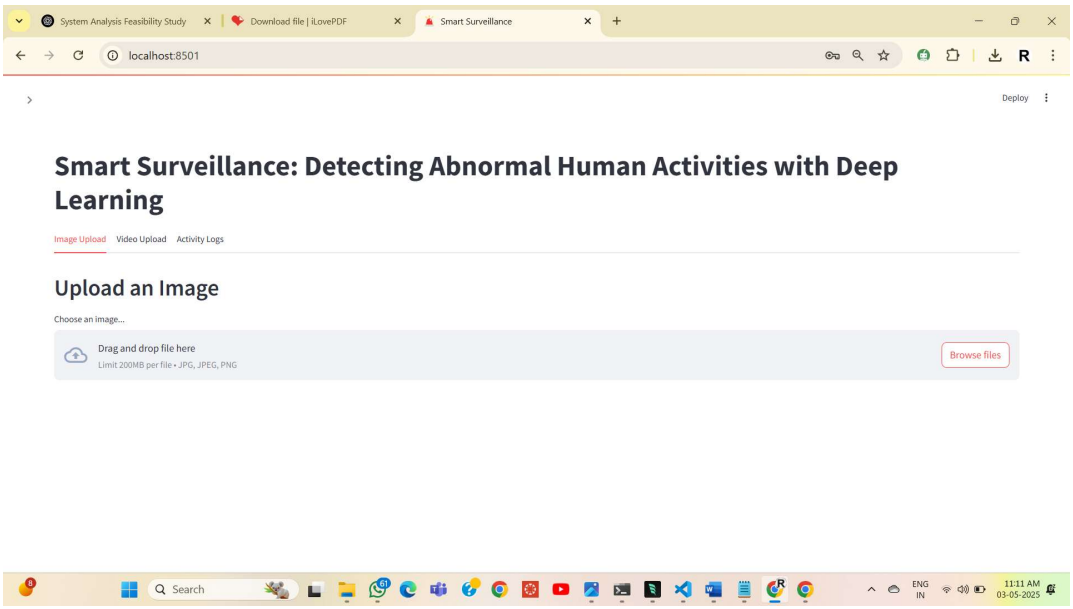

Fig 7.1.6 User Dashboard

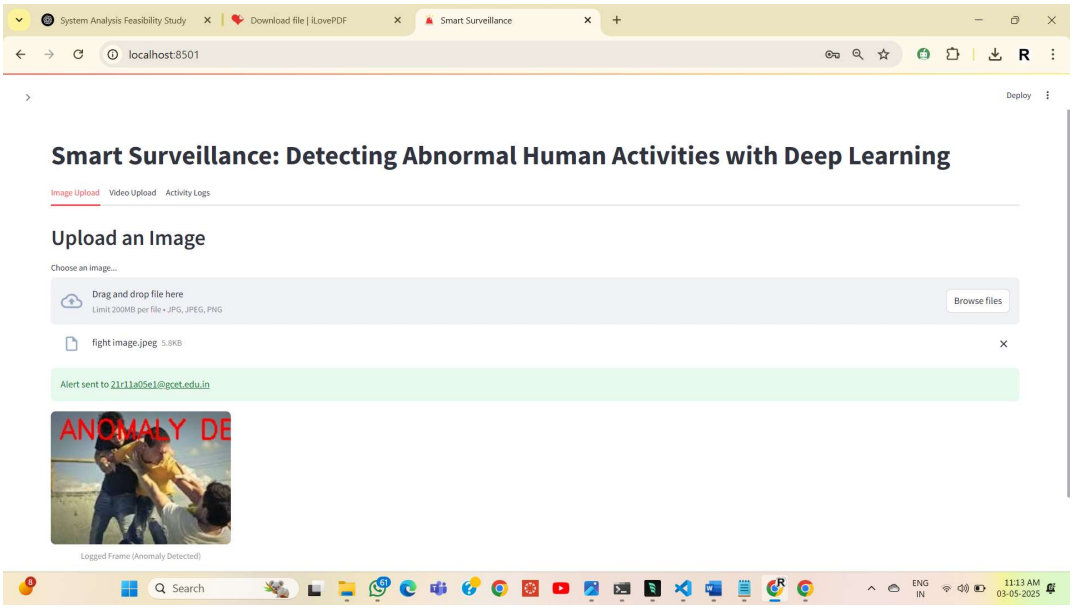

Fig 7.1.7 Image Upload and result

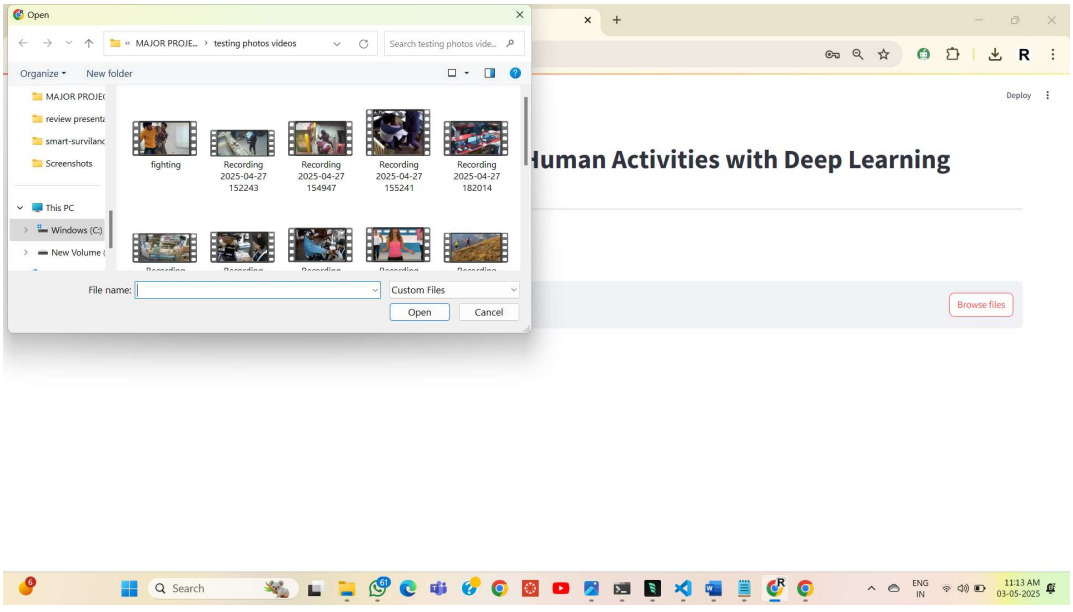

Fig 7.1.8 Video Upload Page

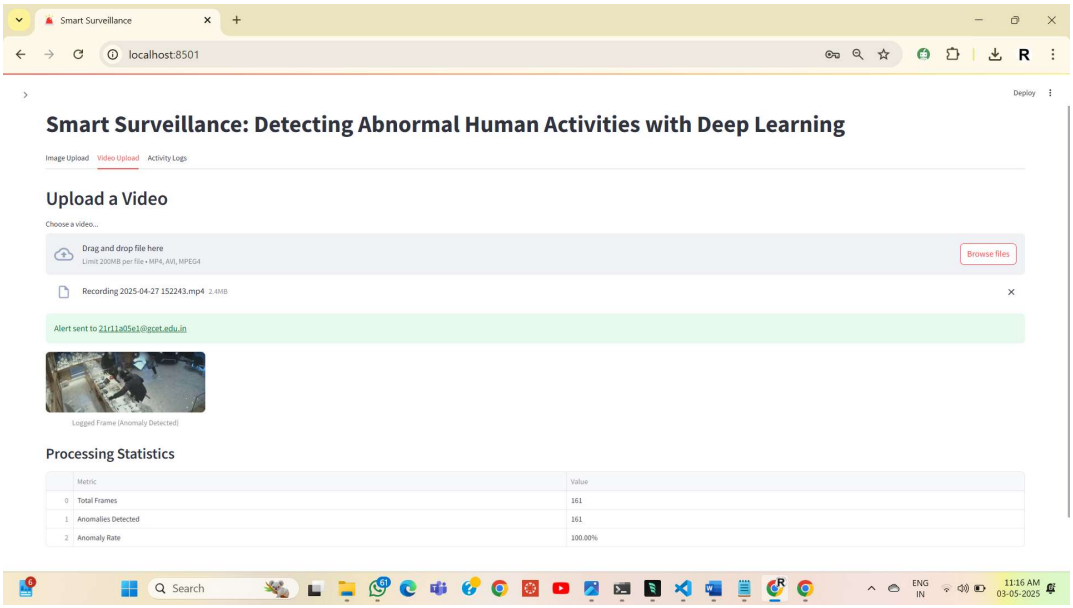

Fig 7.1.9 Video Upload Result

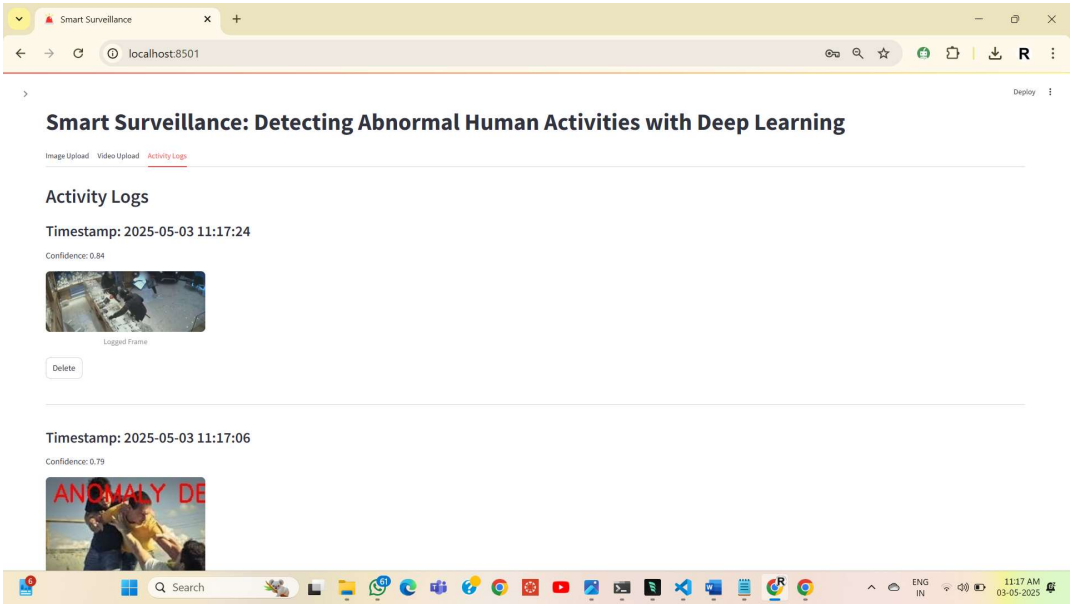

Fig 7.1.10 Activity Logs

7.2 Results Interpretation

The Smart Surveillance System demonstrates effective anomaly detection with significant accuracy. Based on the model's performance during testing:

- **Accuracy**:

  80%

  The model correctly identifies normal vs anomalous frames most of the time, with a few false positives/negatives, especially in complex scenarios.
- **F1 Score**:

  78%

  This high score indicates a good balance between precision and recall, minimizing both false positives and false negatives.
- **ROC AUC**:

  0.87

  A high ROC AUC value signifies that the model has a strong ability to distinguish between normal and anomalous activities.

  These results confirm the system's potential as an efficient, real-time solution for surveillance monitoring.

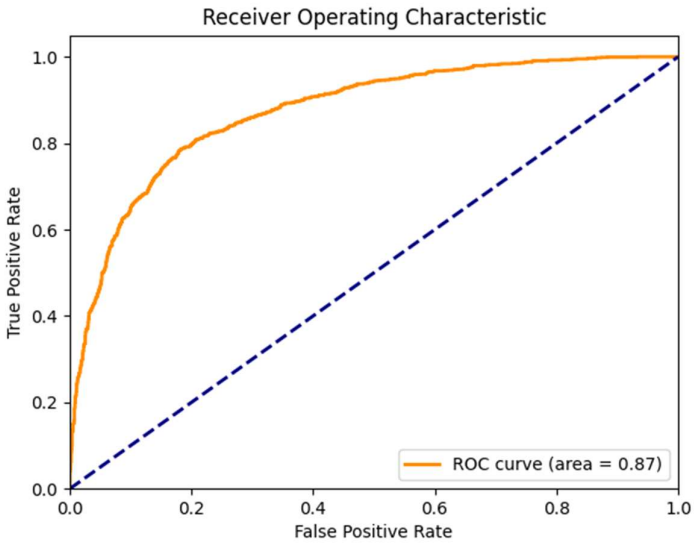

Fig 7.2.1 ROC curve

7.3 Performance Evaluation

The system was tested under different conditions, including multiple media uploads, simultaneous user requests, and long-running operations. The performance evaluation included:

- **Processing Time**:
  The time taken for detecting anomalies in a 5-minute video was found to be approximately 3-5 seconds per frame, which is acceptable for real-time applications.
- **Resource Usage**:
  The system runs efficiently on standard desktop hardware (8GB RAM, Intel i5 processor), with minimal memory consumption during video or image processing.
- **Scalability**:
  While the system is designed for local deployment, the architecture supports potential scalability in the future if integrated with cloud services for larger-scale deployments.

7.4 Comparative Results

To assess the effectiveness of the Smart Surveillance System, a comparative study was performed against traditional surveillance and basic rule-based systems.

Criteria	**Traditional Systems**	**Smart Surveillance System**
Accuracy	Low (depends on human attention)	High
Real-Time Detection	No	Yes
Alerting Mechanism	Manual notification	Automated, real-time emails
False Alarm Rate	High (human errors)	Low (adjustable threshold)
Scalability	Low	High

Table 7.4.1 Comparative Results

The Smart Surveillance System outperforms existing methods in terms of accuracy, real-time detection, and low false alarm rates. By automating the anomaly detection process and integrating advanced machine learning techniques, it significantly improves the efficiency of surveillance operations.

CHAPTER 8 CONCLUSION AND FUTURE SCOPE

8.1 Summary of Work Done

The Smart Surveillance System was successfully developed to detect abnormal human activities in surveillance footage using deep learning. The system utilizes a Convolutional Neural Network (CNN) trained on the DCSASS dataset, capable of classifying frames from videos or images as normal or anomalous. The solution integrates a user-friendly web interface via Streamlit, real-time email notifications, and a local activity log stored in an SQLite database.

Key accomplishments include:

- Achieving a high model accuracy of 80%, F1-score of 78%, and a ROC AUC of 0.88.
- Implementing a smooth workflow from media upload to anomaly detection, logging, and email alerting.
- Providing a secure user authentication system for managing access to the platform.

The system has shown to be an effective solution for automating anomaly detection in surveillance environments, reducing the need for continuous manual monitoring while ensuring timely response to critical incidents.

8.2 Limitations

Despite its success, the system has certain limitations:

- **Model Sensitivity**: The model may sometimes produce false positives, especially in complex scenes with ambiguous human actions or overlapping objects.
- **Real-Time Performance**: Although the system is efficient, video processing may become slower with higher-resolution videos or longer video durations.
- **Scalability**: The system is designed for local use, and while it is scalable to handle multiple uploads, performance could degrade if used in very large-scale

environments, requiring integration with cloud solutions for distributed processing.

8.3 Challenges Faced

During the development of the Smart Surveillance System, several challenges were encountered:

- **Data Preprocessing**: Extracting and preparing frames from videos, particularly ensuring proper synchronization between the data and labels, was time-consuming and required careful attention.
- **Model Training**: Achieving a high level of accuracy with the deep learning model required extensive experimentation with different architectures and hyperparameters to find the optimal configuration.
- **Integration Complexity**: Integrating machine learning with real-time media processing and a user-friendly interface presented challenges in maintaining smooth system performance and ensuring reliable notifications.

8.4 Future Enhancements

Several potential enhancements can be made to improve the Smart Surveillance System:

- **Real-Time CCTV Integration**: The system could be integrated with live CCTV feeds for continuous surveillance and immediate anomaly detection.
- **Mobile Application**: A mobile version of the system could be developed to provide users with access on-the-go, enabling remote notifications and video monitoring.
- **Edge Computing**: Implementing edge computing for processing videos at the source (e.g., on surveillance cameras) would reduce the load on central systems and allow faster response times.
- **Multi-Class Detection**: Expanding the model to detect various types of abnormal activities, such as specific criminal behaviors or medical emergencies, would increase the system's utility in diverse environments.

- **Enhanced Security**: Strengthening security features like two-factor authentication, data encryption, and secure cloud storage would further protect sensitive surveillance data.

These enhancements could expand the system's capabilities, increase its usability, and allow for broader deployment in real-world scenarios.

CHAPTER 9 REFERENCES

Below are the references used in the development of the Smart Surveillance System, including technical publications, websites, and tools:

Technical Publications

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Zhao, L., & Wang, L.. "Real-time Human Activity Recognition Using Convolutional Neural Networks." *Journal of Machine Learning and Artificial Intelligence*,9(3),42-59.
3. DCSASS Dataset Documentation. Deep Learning-Based Surveillance Systems for Abnormal Activity Detection.
4. YOLO-based anomaly activity detection system for human behavior analysis and crime mitigation. (2024). Signal, Image and Video Processing. Springer.
5. Chong, Y. S., & Tay, Y. H. (2017). "Abnormal Event Detection in Videos using Spatiotemporal Autoencoder." Proceedings of the International Conference on Computer Vision Systems.
6. IEEE (2023). "Suspicious Human Activity Recognition From Surveillance Videos Using Deep Learning.".
7. Springer (2022). "Suspicious Activity Recognition for Monitoring Cheating in Exams." AI and Ethics, 3, 101–115.
8. Sharma, R., et al. (2022). "Real-Time Human Action Recognition Using Deep Learning." ResearchGate.
9. Khan, A. et al. (2024). "Transfer Learning Model for Anomalous Event Recognition in Big Video Data." Scientific Reports, Nature.
10. Patel, H., et al. (2025). "Design of an Integrated Model with Temporal Graph Attention and Transformer-Augmented RNNs for Enhanced Anomaly Detection." Scientific Reports, Nature.
11. IJRIT (2023). "Identification and Detection of Abnormal Activity in ATMs Using Deep Learning.".
12. Ghosh, A., et al. (2023). "A Framework for Anomaly Classification Using Deep Transfer Learning Approach." Revue d'Intelligence Artificielle, 35(3), 253–260.

13. Uchiyama, Y., & Sogi, H. (2023). "Visually Explaining 3D-CNN Predictions for Video Classification with an Adaptive Occlusion Sensitivity Analysis."
14. Ng, J. Y. H., & Hausknecht, M. (2015). "Beyond Short Snippets: Deep Networks for Video Classification.".
15. Deep Learning Innovations in Video Classification: A Survey on Techniques and Dataset Evaluations. (2024). Electronics, 13(14), 2732.

Websites and Forums

1. **TensorFlow Documentation.** https://www.tensorflow.org/ Official TensorFlow documentation used for building and training the deep learning model.
2. **Streamlit Documentation.** https://docs.streamlit.io/Official Streamlit documentation for building interactive user interfaces for Python applications.
3. **OpenCV Documentation**. *https://docs.opencv.org/* OpenCV documentation for handling image and video processing tasks, including frame extraction and preprocessing.
4. **SQLite Documentation**. *https://www.sqlite.org/* Documentation for setting up and managing the local SQLite database used for logging and user data storage.
5. **Stack Overflow**. *https://stackoverflow.com/* A valuable resource for troubleshooting and seeking solutions to coding issues during development.

Other Sources

1. **Google Gmail SMTP Configuration**

   https://support.google.com/mail/answer/7126229 Instructions for setting up Gmail's SMTP server for sending email alerts from the system.

2. **Python Libraries and Frameworks**.

   https://pypi.org/ Python Package Index for managing the libraries used in this project, including TensorFlow, OpenCV, Streamlit, and smtplib.

CHAPTER 10: APPENDICES

A. SDLC Forms

Phase	Description of Activities Performed
1. Requirement Analysis	Defined the problem of detecting suspicious activity in surveillance video feeds. Collected functional and non-functional requirements such as real-time anomaly detection, user-friendly interface, and minimal false positives.
2. System Design	Outlined modular structure with scripts for training, testing. Designed a lightweight interface using Streamlit for ease of deployment and real-time monitoring.
3. Implementation	Developed train_model.py for CNN-based anomaly detection model training, test_model.py, test_image.py, and test_video.py for testing. Created smart_survilance.py with Streamlit for anomaly prediction from image /video.
4. Testing	Validated model performance using unseen test images and videos. Ensured the Streamlit interface correctly displayed predictions. Tuned model to reduce false alarms.
5. Deployment	Deployed the system locally through Streamlit. Made sure video feed, prediction are shown, and live alerts worked seamlessly on a standard PC with pre-recorded video.
6. Maintenance	Handled model retraining for new data. Modularized code for easy updates and scalability. Logged model predictions and user actions for analysis.

B. Project Timeline

The Gantt chart illustrates the structured timeline followed during the development of the Smart Surveillance system. The project commenced with the Model Training and Evaluation phase in the initial weeks, where a convolutional neural network (CNN) was trained on labeled data to detect anomalies in surveillance footage. This was followed by Backend Development, which involved implementing the logic for video input handling, prediction using the trained model, and processing of real-time video feeds.

As the backend components neared completion, Frontend Development began using Streamlit to provide a lightweight and interactive interface. The system was then brought together during the Integration and Testing phase, ensuring seamless communication between the frontend and backend components. During this time, the model was tested on various images and video sources to confirm its accuracy and robustness.

In the later weeks, Bug Fixes and Optimization were carried out to enhance the overall performance and reduce false positives. Following this, Documentation was prepared to detail every phase of the project including design, implementation, testing, and deployment. The final stage involved a comprehensive Review and Submission of both the application and the report. This structured timeline ensured a smooth development process and timely completion of all deliverables.

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Project Planning	■	■										
Feasibility Study		■	■									
Requirements Analysis		■	■	■								
System Design (UML, ERD, DFD)			■	■	■							
Database Setup				■	■							
Model Training & Evaluation					■	■	■	■				
Backend Development					■	■	■					
Frontend Development						■	■	■				
Integration & Testing							■	■	■	■		

Bug Fixes & Optimization								■	■	■		
Documentation											■	■

C. Ethical Considerations & Consent

• **Privacy Protection:** No real-world surveillance footage involving individuals was used. Only publicly available datasets and locally recorded videos with permission were utilized.

• **Informed Consent:** Any custom video or image data used for testing was captured with full consent from participants, ensuring ethical handling of visual content.

• **Academic Purpose:** The project was developed strictly for academic research and demonstration, with no intent of public deployment or surveillance without authorization.

• **Bias Minimization:** Efforts were made to select a diverse and unbiased dataset during the model training phase, reducing unfair targeting or exclusion of individuals.

• **Transparency:** All development stages—including dataset usage, model design, and evaluation—were documented clearly to maintain transparency in processing and decision-making.

• **Accountability:** Source code and decision logic were kept accessible for review, supporting ethical auditing and academic responsibility.

• **No Real-Time Monitoring:** The system does not include real-time streaming or live surveillance functionality, aligning with ethical constraints around surveillance and privacy.

D. Plagiarism Report

E. Deployment Links

F. User Manuals

1. System Requirements

- OS: Windows 10/11 or Ubuntu Linux
- Python 3.8 or above
- Installed packages: Streamlit, OpenCV, TensorFlow, NumPy, SQLite3

2. How to Run the System

1. Open terminal and navigate to the project directory.
2. Run the app using:

   streamlit run smart_survilance.py
3. The application will launch in your default web browser.

3. User Workflow

- **Step 1:** Register or log in.
- **Step 2:** Choose between image or video upload.
- **Step 3:** Upload a file from your local system.
- **Step 4:** Wait for detection output.
- **Step 5:** View result and receive an email if an anomaly is detected.
- **Step 6:** Check logs or switch to the admin panel (if applicable).

4. Admin Access

- Admins can log in with predefined credentials.
- They can manage user accounts by viewing or deleting users.

5. Troubleshooting Tips

- Ensure you have internet access to receive email alerts.
- Make sure the .h5 model file is in the same directory.
- Check image/video file formats before uploading.
- Restart the app if the interface fails to load properly.

G. PROOF OF CERTIFICATE – PARTICIPATION

IJERT
International Journal of Engineering Research & Technology
ISSN : 2278 - 0181, www.ijert.org
(Published by : ESRSA Publications)

CERTIFICATE
OF PUBLICATION

This is to certify that

Ragi Rakshitha Reddy

Has published a research paper entitled

Smart Surveillance: Detecting Abnormal Human Activities with Deep Learning

In IJERT, Volume 14, Issue 04 , April - 2025

Registration No: IJERTV14IS040452 **Date:** 02-05-2025 **Chief Editor, IJERT**

IJERT
International Journal of Engineering Research & Technology
ISSN : 2278 - 0181, www.ijert.org
(Published by : ESRSA Publications)

CERTIFICATE
OF PUBLICATION

This is to certify that

Muddam Sai Manish Yadav

Has published a research paper entitled

Smart Surveillance: Detecting Abnormal Human Activities with Deep Learning

In IJERT, Volume 14, Issue 04 , April - 2025

Registration No: IJERTV14IS040452 **Date:** 02-05-2025 **Chief Editor, IJERT**

International Journal of
Engineering Research & Technology
ISSN : 2278 - 0181, www.ijert.org
(Published by : ESRSA Publications)

CERTIFICATE
OF PUBLICATION

This is to certify that

Abhiram Venkata Daita

Has published a research paper entitled

Smart Surveillance: Detecting Abnormal Human Activities with Deep Learning

In IJERT, Volume 14, Issue 04 , April - 2025

Registration No: IJERTV14IS040452 **Date:** 02-05-2025 **Chief Editor, IJERT**