“Suspicious Activity Simulator”

Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF SCIENCE (B.Sc)



*Submitted by:*

Dishant Uprety(222MCS17)

Under the guidance of

Ms Mary Merline Rani C

Ms Pooja A

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

St Joseph’s University

Lalbagh Road, Bengaluru-560027

DEPARTMENT OF COMPUTER SCIENCE



CERTIFICATE OF COMPLETION

This is to certify that the project entitled “Suspicious Activity Simulator” has been satisfactorily completed by Dishant Uprety(222MCS17 partial fulfilment of the award of the Bachelor of Science degree requirements prescribed by St Joseph’s University Bengaluru during the academic year 2024 - 2025.

Guide Signature Head of the Department

ACKNOWLEDGEMENT

The success of the project depends upon the efforts invested. It’s my duty to acknowledge and thank the individuals who has contributed in the successful completion of the project.

I take this opportunity to express my profound and whole hearted thanks to Rev. Fr. Dr. VICTOR LOBO SJ, Vice Chancellor, St. Joseph’s University, Bangalore and Rev.Fr. DENZIL LOBO SJ, Director, St. Joseph’s Institute of Information Technology (SJIIT) for providing ample facilities made to undergo our project successfully.

I express my deep sense of gratitude and sincere thanks to our Head of the Department Dr. B.G Prasanthi for her valuable advice.

I feel immense pleasure to thank my respected guides Prof Sara Kutty and Prof Vineetha for sustaining Interest and providing dynamic guidance in aiding me to complete this project immaculately and impeccably and for being the source of my strength and confidence.

It is my duty to express my thanks to all Teaching and Non-Teaching Staff members of Computer science department who offered me help directly or indirectly by their suggestions. The successful completion of my project would not have been possible without my parent’s Sacrifice, guidance and prayers. I take this opportunity to thank everyone for their continuous Encouragement. I convey my thankfulness to all my friends who were with me to share my happiness and agony. Last but not the least I thank almighty God for giving me strength and good health throughout my project and enabling me to complete it successfully.

DECLARATION

We, Dishant Uprety, Muzakkir Hussain, Nihal Shukla and Aryan Kujur, hereby declare that the project work entitled “Suspicious Activity Simulator” for Computer Science department is an original project work carried out by us, under the guidance of *Ms Mary Merline Rani C ,Ms Pooja A and Ms saryana M .* This project work has not been submitted earlier either to any University / Institution or any other body for the fulfilment of the requirement of a course of study.

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S No. | Topic | Page No. |
| 1 | Abstract | 1 |
| 2 | Introduction | 2 |
| 2.1 | Problem definition | 3 |
| 2.2 | Scope of the project | 4-6 |
| 2.3 | Objectives | 6-8 |
| 3 | Requirements specification | 8 |
| 3.1 | Hardware | 8 |
| 3.2 | Software | 9 |
| 4 | Development tools / technology | 9-11 |
| 5 | Study of Existing system | 11-13 |
| 6 | Analysis | 14 |
| 6.1 | Feasibility | 14 |
| 6.1.1 | Technical Feasibility | 14 |
| 6.1.2 | Operational Feasibility | 15 |
| 6.1.3 | Economic Feasibility | 15-16 |
| 7 | Requirement Analysis | 16-18 |
| 8 | System Design | 18-32 |
| 8.1 | ER diagram | 32-34 |
| 8.2 | Data dictionary | 35-36 |
| 8.3 | User design | 36 |
| 8.4 | Admin design | 37-38 |
| 8.5 | Table design | 39 |
| 9 | Sample code | 40-62 |

|  |  |  |
| --- | --- | --- |
| 10 | Implementation | 62-63 |
| 11 | Planning Scheduling | 63-68 |
| 12 | Future Enhancement | 68 |
| 13. | Summary and Conclusion | 69-71 |
| 14. | Reference | 71 |

**ABSTRACT**

The “Suspicious Activity Simulator” is an advanced surveillance system designed to detect and classify suspicious human activities in real-time using machine learning and deep learning techniques. As security threats continue to rise, traditional surveillance systems face limitations due to their reliance on manual monitoring, which can be inefficient, error-prone, and time-consuming. This project aims to bridge that gap by automating threat detection through artificial intelligence, significantly enhancing response time, accuracy, and situational awareness for security personnel.

The system leverages pre-trained deep learning models such as SSD MobileNet V2 COCO 2018 for real-time human detection and a custom-trained CNN model (modelnew.h5) for violence detection. The violence detection model was developed using a dataset comprising 350 videos depicting violent actions and 350 videos representing non-violent activities, ensuring a well-balanced dataset for robust classification. The project integrates real-time video processing using OpenCV and provides a CustomTkinter-based GUI for an intuitive user experience, allowing users to monitor video feeds and detection results seamlessly.

Upon detecting suspicious activities, the system immediately triggers an alert mechanism by sending a real-time message to a designated Telegram group. This ensures that threats are identified and addressed promptly, reducing potential harm and improving safety measures. The system is designed to be adaptable and scalable, making it suitable for various environments such as public spaces, shopping malls, corporate offices, transportation hubs, and smart city infrastructure.

Future enhancements include multi-camera integration to monitor larger areas simultaneously, behavioral pattern analysis using advanced AI techniques for anomaly detection, edge computing deployment to reduce processing latency, and cloud-based storage for managing large-scale surveillance data efficiently. By incorporating these enhancements, the system can evolve into a fully autonomous and intelligent security solution.

With its real-time processing, high accuracy, and automated alert system, the Suspicious Activity Simulator has the potential to revolutionize the field of automated security surveillance, reducing dependency on manual supervision while significantly enhancing overall safety and security in critical areas.

**INTRODUCTION**

Security threats and suspicious activities are growing concerns in today’s world, with rising crime rates and increasing risks in public and private spaces. From unauthorized access in restricted areas to violent incidents in crowded places, security agencies face significant challenges in detecting and responding to suspicious activities in real time. According to a report by the United Nations Office on Drugs and Crime (UNODC), global crime rates have been increasing, with violent crimes and theft being among the most common security concerns. Additionally, research by the Urban Institute suggests that surveillance systems can reduce crime by up to 51% in some areas when properly implemented. However, traditional surveillance systems rely heavily on human monitoring, which is prone to fatigue, errors, and delayed response times.

To address this issue, the Suspicious Activity Simulator aims to bridge the gap between security monitoring and real-time threat detection using machine learning and computer vision techniques. This project is designed to simulate, detect, and analyze suspicious behavior automatically, reducing human effort and improving the efficiency of surveillance systems. By leveraging artificial intelligence (AI), the system can identify anomalous activities such as sudden aggressive movements.

The simulator integrates computer vision algorithms, motion tracking, and anomaly detection techniques to analyze live video feeds and prerecorded videos or feeds , it identifies activities that pose a security threat. This system can automatically flag suspicious behavior and generate real-time alerts, ensuring a faster response time. Additionally, the project includes a GUI based, making it easy for security personnel to interact with and monitor the system effectively.

The Suspicious Activity Simulator is designed to tackle these challenges by offering an automated, AI-driven approach to suspicious activity detection. The system’s ability to process live video feeds and prerecorded feeds , detect anomalies, and provide real-time alerts makes it a valuable tool.

By integrating machine learning with an interactive, user-friendly GUI, this project makes security surveillance smarter, faster, and more reliable. Ultimately, the Suspicious Activity Simulator is a step toward a safer and more secure future, where AI enhances human decision-making and prevents threats before they escalate.

**Problem definition**

Traditional security surveillance systems rely heavily on human monitoring, which is prone to fatigue, inefficiency, and delayed responses. In environments such as public spaces, offices, and transportation hubs, quickly identifying suspicious behavior is crucial for preventing security incidents. However, manual monitoring is highly error-prone, and reviewing footage after an incident occurs is a reactive approach rather than a preventive measure. Existing surveillance solutions lack automated intelligence, making it challenging to detect anomalies like loitering, sudden aggressive movements, or abandoned objects without constant human supervision. Security teams often spend hours manually analyzing footage, resulting in slow response times and increased operational costs.

The limitations of conventional surveillance systems highlight the urgent need for AI-driven automation. Human efficiency in monitoring declines over time, leading to missed critical events. Most systems merely record footage rather than proactively detecting threats in real time. Additionally, manually reviewing vast amounts of video data is an arduous and time-consuming process that hampers effective security responses. The absence of intelligent automation further restricts the ability of traditional surveillance setups to recognize patterns and detect anomalies efficiently. Given these challenges, implementing an AI-powered system that integrates real-time detection, analysis, and alert mechanisms is essential for enhancing security and ensuring a more proactive approach to threat prevention.

**SCOPE OF THE PROJECT**

Below is a comprehensive project scope for a real-time violence detection system using machine learning. This documentation is designed to be highly detailed and technical, covering multiple aspects ranging from conceptual design to real-world implementation challenges.

The primary objective of this project is to develop a robust real-time violence detection system that leverages state-of-the-art machine learning algorithms to analyze video streams and detect potentially violent behavior with high accuracy. This system aims to process live video feeds from various sources, such as CCTV cameras or mobile devices, and provide prompt alerts when suspicious activities are detected. The project is deeply rooted in computer vision and pattern recognition, employing advanced techniques such as convolutional neural networks (CNNs). In designing the system, one of the critical challenges is ensuring that the detection process runs with minimal latency while maintaining high detection accuracy. This necessitates the integration of efficient algorithms and the optimization of neural network architectures for real-time inference, possibly through the use of edge computing devices and hardware accelerators like CPUs/GPUs/TPUs.

The system is built on a modular architecture, where each module is responsible for distinct tasks such as video frame acquisition, pre-processing, feature extraction classification, and alert generation. The pre-processing stage involves tasks like video stabilization, noise reduction, and frame normalization, which are critical to ensuring that the input data is of sufficient quality for analysis. Feature extraction will heavily rely on deep learning models that are pre-trained on datasets, which are then fine-tuned on violence-specific data. This process involves the development of custom architecture tailored to capture the nuanced motions and interactions that precede violent incidents. The classifier, which forms the heart of the system, will operate on the extracted features to make real-time predictions. Given the inherent complexity of real-time video data, multiple strategies such as sliding window approaches, ensemble methods, and probabilistic fusion of temporal and spatial information will be evaluated to optimize performance.

Another major focus of the project is ensuring the system’s scalability and adaptability in various operational environments. The software framework will be designed with a microservices architecture, facilitating seamless integration with existing security infrastructure. This approach not only improves maintainability and scalability but also allows for continuous updates and integration of new detection models as they are developed. The deployment strategy will encompass both cloud-based solutions for centralized monitoring and on-premises edge devices for critical, latency-sensitive applications. Additionally, the system is expected to interface with other safety and security protocols, integrating with law enforcement alert systems and real-time incident reporting dashboards. The deployment phase will involve rigorous testing in simulated environments followed by pilot implementations in controlled settings to validate system performance under real-world conditions.

From a data perspective, this project will require the assembly and curation of extensive datasets that encapsulate a broad spectrum of violent and non-violent behaviors. This involves sourcing data from public repositories, as well as collaborating with academic institutions and law enforcement agencies to obtain annotated video footage. The data curation process is critical, as it must address issues such as class imbalance, diverse environmental conditions, and variations in camera quality. Techniques such as data augmentation, synthetic data generation, and domain adaptation will be employed to mitigate these challenges. Moreover, the system will incorporate online learning components to adapt to new patterns of behavior, making it a continuously evolving system that improves with time. Ethical considerations and privacy-preserving measures are paramount, requiring the system to comply with regulations like GDPR while ensuring that data is anonymized and securely stored.

The research and development component of the project will delve into various cutting-edge topics. These include the exploration of spatio-temporal feature representations, optimization of deep learning inference on constrained hardware, and the use of explainable AI (XAI) techniques to interpret the decision-making process of the violence detection system. The integration of XAI is particularly significant as it provides transparency in how decisions are made, thereby increasing trust among end-users and stakeholders. The project will also investigate the use of unsupervised and semi-supervised learning techniques to reduce dependency on large labeled datasets, which is often a limiting factor in specialized domains like violence detection. Furthermore, continuous benchmarking against standard datasets and participation in academic challenges will be integral to validating the performance and robustness of the developed models.

In summary, the real-time violence detection system using machine learning is a multidisciplinary project that spans deep learning, computer vision, systems engineering, and data science. It requires an intricate balance between algorithmic efficiency, system scalability, and ethical data management. The project’s scope is comprehensive, covering everything from initial design and dataset curation to model development, system integration, and deployment. By pushing the boundaries of current ML technologies and leveraging state-of-the-art research, this project not only aims to enhance public safety but also contributes significantly to the academic and industrial discourse on real-time anomaly detection in video streams.

This detailed scope should provide a solid foundation for your project documentation and help communicate the depth and breadth of your research to both technical and non-technical stakeholders.

**OBJECTIVES OF THE SUSPICIOUS ACTIVITY SIMULATOR**

The *Suspicious Activity Simulator* is a machine learning-based detection system designed to identify and classify suspicious activities with high accuracy. It is trained using datasets containing both normal and abnormal behaviors, allowing it to distinguish between safe and potentially dangerous actions. Depending on data availability and the type of anomalies being detected, it utilizes supervised learning techniques to enhance its effectiveness. The core functionality of this detection model is built using deep learning frameworks such as TensorFlow, ensuring robust and scalable performance.

To achieve precise detection, the project integrates advanced computer vision techniques for processing video frames and analyzing human movements. It applies object detection, motion tracking, and feature extraction algorithms, enabling the system to identify unusual behaviors in real-time. Tools such as OpenCV are utilized for image processing, while deep learning models enhance accuracy in detecting suspicious activity patterns. These techniques significantly improve the reliability of the system across diverse surveillance environments.

A key aspect of the project is the simulation of suspicious activities, allowing controlled testing of different security threats without relying solely on real-world surveillance footage. The simulator generates or processes synthetic scenarios to assess the model's response to various activities, such as violence .

The system enhances security by proactively detecting threats before they escalate. By providing real-time alerts, it assists in preventing crimes such as, violent incidents by immediately notifying security personnel. The simulator’s adaptability makes it suitable for deployment in public spaces, office buildings, transportation hubs, and other high-risk environments where safety is a priority. Integration with existing security infrastructure allows seamless operation and improves overall situational awareness.

To make the system accessible to a wider audience, an interactive graphical user interface (GUI) is developed using CustomTkinter. This interface allows users to load video feeds, monitor real-time detections, configure settings, and review logs of suspicious activities. Designed for usability, the GUI ensures that even individuals without technical expertise can operate and interpret the system’s functionalities effectively. This feature enhances the practical deployment of the simulator across various industries, including law enforcement and private security firms.

One of the significant challenges in activity recognition is minimizing false positives and false negatives, ensuring that the system accurately identifies violence while avoiding unnecessary alarms. To address this, the project implements optimization techniques such as hyperparameter tuning, feature selection, and real-time data augmentation. These improvements enhance detection accuracy while ensuring that the system processes video streams efficiently, making it suitable for real-time applications where rapid response is crucial.

The simulator is designed for real-time monitoring, capable of processing live video streams and detecting suspicious activities as they occur. Immediate alerts are generated upon detecting anomalies, allowing security teams to respond promptly. It includes SMS notifications that integrates with existing security systems to enable automated threat response mechanisms. This further improves the system’s effectiveness in high-security areas, making it a valuable addition to modern surveillance infrastructure.

Beyond practical security applications, this simulator serves as a research and training tool for law enforcement agencies, security professionals, and academic researchers. By analyzing patterns of suspicious activities, it helps improve surveillance techniques and train personnel in violence recognition. Researchers also benefit from using the simulator to develop and test new machine learning models, contributing to advancements in security analytics and automated surveillance technologies.

**REQUIREMENTS SPECIFICATION**

**HARDWARE REQUIREMENTS**

To ensure efficient processing, real-time analysis, and smooth execution of the Suspicious Activity Simulator, the following hardware components are required:

Processor: Intel Core i5/i7 with at least 4 cores.

RAM: Minimum 8GB

Storage: 256GB SSD

GPU (Optional but Recommended): NVIDIA GTX 1650 or higher (for deep learning-based analysis, if applicable)

Camera: HD surveillance cameras (or webcam for testing) with at least 720p resolution

Power Supply: Stable power source to prevent disruptions in surveillance

**SOFTWARE REQUIREMENTS**

The software components ensure the seamless operation of the Suspicious Activity Simulator, including AI-based detection and GUI functionality.

Operating System: Windows 10/11

Programming Language: Python 3.x

Machine Learning Libraries:

OpenCV (for image/video processing)

TensorFlow/Keras or Scikit-Learn (machine learning model implementation)

NumPy, Pandas (for data handling)

GUI Framework: CustomTkinter (for interactive interface)

Database (if required for logging activities): SQLite/MySQL

Other Dependencies:

Matplotlib (for visualization)

Pillow (for image processing)

PyTorch (for deep learning models)

**DEVELOPMENT TOOLS / TECHNOLOGIES**

The Suspicious Activity Simulator utilizes a combination of machine learning, deep learning, and computer vision technologies to detect and classify suspicious activities in real time. The development tools and technologies used in this project include:

**PROGRAMMING LANGUAGES & FRAMEWORKS**

Python: Primary language for model development and GUI implementation.

CustomTkinter: Used for developing the GUI interface.

OpenCV: Used for real-time video processing and frame extraction.

TensorFlow / Keras: Used for training and deploying deep learning models.

**MACHINE LEARNING MODELS & PRETRAINED MODELS**

Human Detection: The system uses a pretrained SSD MobileNet V2 model (ssd\_mobilenet\_v2\_coco\_2018\_03\_29) for detecting humans in video streams efficiently.

Violence Detection: A custom deep learning model (modelnew.h5) was trained using 350 violence videos and 350 non-violence videos to classify violent activities.

Feature Extraction: Used Convolutional Neural Networks (CNNs) to extract features from video frames for activity classification.

**DEVELOPMENT & DEPLOYMENT TOOLS**

VSCode: Used for model development and testing.

NumPy & Pandas: Used for data preprocessing and manipulation.

Matplotlib & Seaborn: Used for visualizing model training results.

Scikit-learn: Used for evaluating model performance and tuning hyperparameters.

Telegram API: Used for sending real-time alerts in case of suspicious activity detection.

**DATABASE & STORAGE**

SQLite / MySQL: Used for storing activity logs, detected events, and alert history.

Local Storage: Used to store trained models and log files.

These tools and technologies enable the Suspicious Activity Simulator to process video feeds in real-time, detect potential threats, and alert security personnel efficiently.

**STUDY OF EXISTING SYSTEMS**

Traditional Surveillance Systems (CCTV-Based Monitoring)

Most security systems today rely on CCTV cameras with human monitoring. These systems record video footage, which is later reviewed in case of incidents.

**HOW IT WORKS:**

CCTV cameras capture and store footage.

Security personnel manually monitor live feeds or review recorded video after an event.

Some systems allow motion detection, but they lack intelligent decision-making.

**LIMITATIONS:**

Human Monitoring Fatigue – Security personnel cannot maintain focus for extended periods.

Delayed Response – Crimes are often detected after they occur instead of in real-time.

No AI-Based Analysis – Cameras do not recognize suspicious activities or behavioral patterns.

High Storage-requirements large amounts of video footage need to be stored and managed.

**MOTION DETECTION & OBJECT TRACKING SYSTEMS**

Some security cameras feature motion detection to alert security teams when movement is detected. These systems use basic computer vision techniques but lack advanced AI-based behavior analysis.

**HOW IT WORKS:**

Detects movement and triggers an alert.

Some systems track objects but do not differentiate between normal and suspicious activities.

**LIMITATIONS:**

False Positives – Detects all motion, including non-suspicious activities (e.g., passing pedestrians).

Lack of Context Awareness – Cannot analyze behaviors such as loitering or aggressive movements.

Basic Threshold-Based Alerts – Does not use machine learning to improve accuracy.

**AI-POWERED SURVEILLANCE SYSTEMS (HIGH-END COMMERCIAL SOLUTIONS)**

Advanced security companies offer AI-driven surveillance solutions that analyze human behavior using deep learning and anomaly detection models.

**FEATURES:**

Facial recognition and behavior analysis.

Machine learning models detect suspicious movements.

Automated alerts with real-time threat detection.

**LIMITATIONS:**

Expensive – High-end AI surveillance solutions are costly and require powerful hardware.

Privacy Concerns – Many systems use facial recognition, raising ethical and legal issues.

Complex Setup – Requires specialized infrastructure and technical expertise.



**ANALYSIS & FEASIBILITY STUDY FOR SUSPICIOUS ACTIVITY SIMULATOR**

A feasibility study helps determine whether the Suspicious Activity Simulator is practical, cost-effective, and technically viable. This analysis covers the following aspects:

**TECHNICAL FEASIBILITY**

This evaluates whether the required technology and resources are available to develop and deploy the project.

**AVAILABILITY OF TECHNOLOGY**

The project uses Python, OpenCV, and machine learning libraries, which are open-source and well-documented.

The hardware requirements (CCTV cameras, a mid-range CPU) are commonly available.

No need for high-end GPUs, as models can run on a CPU with optimizations.

SOFTWARE & HARDWARE COMPATIBILITY

Works on Windows.

GUI is built with CustomTkinter, ensuring a smooth user experience.

Can be integrated with databases (SQLite/MySQL) for storing logs and alerts.

SCALABILITY & PERFORMANCE

The system can be expanded to support multiple camera feeds.

Optimized machine learning models ensure real-time performance.

**OPERATIONAL FEASIBILITY**

This assesses whether the system will be practical for end-users (security teams, management, etc.) and how effectively it fits into existing workflows.

**USER-FRIENDLY INTERFACE**

The CustomTkinter-based GUI makes monitoring and interaction simple.Security teams can view detected anomalies in real-time without needing technical expertise.

**AUTOMATION & EFFICIENCY**

Reduces reliance on manual surveillance, improving efficiency.

Provides real-time alerts, allowing for faster response times.

**INTEGRATION WITH EXISTING SYSTEMS**

Can work alongside existing CCTV setups without major infrastructure changes.

Low learning curve for operators since it uses familiar interfaces.

Conclusion: The project is operationally feasible, as it enhances surveillance workflows while minimizing manual effort.

**ECONOMIC FEASIBILITY**

This evaluates whether the project is cost-effective compared to existing solutions.

Low Development Costs

Uses open-source tools (Python, OpenCV, TensorFlow, etc.), reducing software costs.

No need for expensive hardware—runs efficiently on standard computers.

Affordable Alternative to Commercial AI Security Systems

AI-powered surveillance solutions from major companies are expensive and require dedicated infrastructure.

The Suspicious Activity Simulator provides similar functionality at a fraction of the cost.

Return on Investment (ROI)

Reduces manpower costs by minimizing the need for constant human surveillance.

Prevents security breaches, potentially saving organizations from financial losses due to theft, vandalism, or threats.

|  |  |  |
| --- | --- | --- |
| Feasibility | Status | Explanation |
| Technical Feasibility | Yes | Uses open source tools, supports real time processing on CPU |
| Operational Feasibility | Yes | Easy to use, integrate with existing camera systems, improves efficiency |
| Economic Feasibility | Yes | Low-cost, reduces manual surveillance expenses, high ROI |

**REQUIREMENT ANALYSIS FOR SUSPICIOUS ACTIVITY SIMULATOR**

Requirement analysis defines the functional and non-functional requirements necessary for developing the Suspicious Activity Simulator. It ensures that the system meets user needs, technical constraints, and project goals.

**FUNCTIONAL REQUIREMENTS**

These define the core features the system must provide.

**SUSPICIOUS** **ACTIVITY** **DETECTION**

Capture live video feeds from a webcam or CCTV camera.

Detect suspicious activities using motion tracking and anomaly detection algorithms.

Classify activities as normal or suspicious based on predefined models.

**REAL-TIME ALERTS & LOGGING**

Generate instant alerts when suspicious activity is detected.

Store detected incidents in a database (SQLite/MySQL) with timestamps.

Allow users to review past detections for analysis.

**GRAPHICAL USER INTERFACE (GUI)**

Provide a CustomTkinter-based GUI for easy interaction.

Display live video feed alongside detection results.

Allow users to configure sensitivity settings for activity detection.

**SYSTEM CONFIGURATION & CUSTOMIZATION**

Users can adjust detection thresholds to reduce false positives.

Option to select specific camera feeds for monitoring.

Ability to export logs for further investigation.

**NON-FUNCTIONAL REQUIREMENTS**

These define the performance, security, and usability aspects of the system.

**PERFORMANCE** **REQUIREMENTS**

Process video in real-time with minimal delay.

Optimize machine learning models for CPU-based execution (since no GPU is available). Ensure low false alarm rates through fine-tuned models.

**SECURITY & PRIVACY REQUIREMENTS**

Restrict access to system settings and logs to authorized users.

Ensure data encryption for stored records if required.

No facial recognition to maintain privacy compliance.

**USABILITY & ACCESSIBILITY**

Intuitive CustomTkinter GUI for non-technical users.

Provide clear notifications when suspicious activity is detected.

Ensure compatibility with Windows Scalability & Maintainability

Support multiple video feeds (if needed in future versions).

Allow easy updates and model improvements.

Maintain modular code structure for future enhancements.

**SYSTEM MODULES OF SUSPICIOUS ACTIVITY SIMULATOR**

The Suspicious Activity Simulator is divided into multiple functional modules, each responsible for a specific task within the system. These modules work together to ensure real-time suspicious activity detection, alert generation, and user interaction. Below is a detailed explanation of each module.

**USER AUTHENTICATION MODULE**

**PURPOSE:**

The User Authentication Module ensures that only authorized personnel can access the system. It prevents unauthorized access by implementing secure login and registration mechanisms.

**FUNCTIONALITIES:**

**USER SIGN-UP:**

* Allows new users (security personnel/admin) to create an account.
* Requires username, password, and a security question for account recovery.
* Encrypts passwords using bcrypt hashing before storing them in the database.

**User Login:**

* Allows registered users to log in with their credentials.
* Validates hashed passwords from the database.
* Redirects to the main dashboard upon successful login.

**USER SESSION MANAGEMENT:**

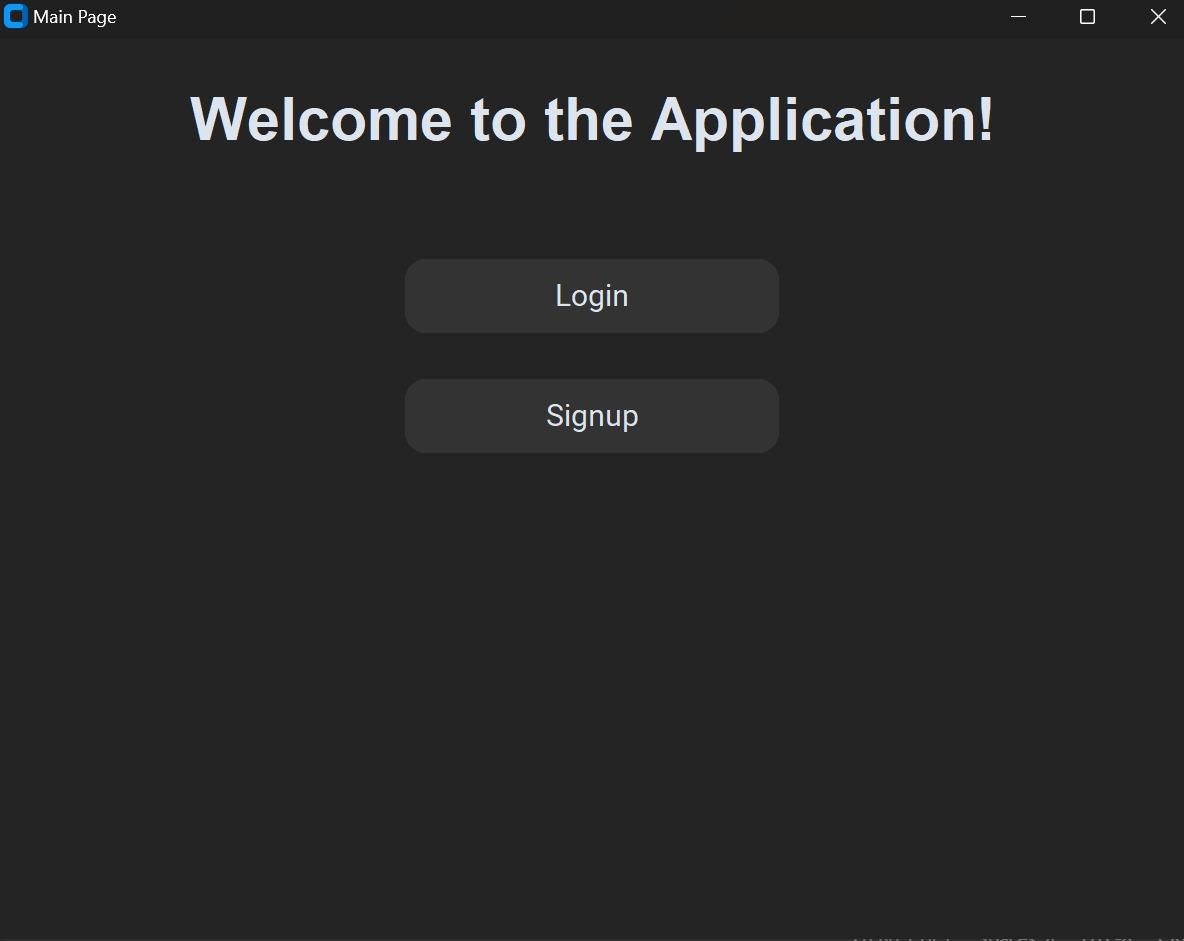
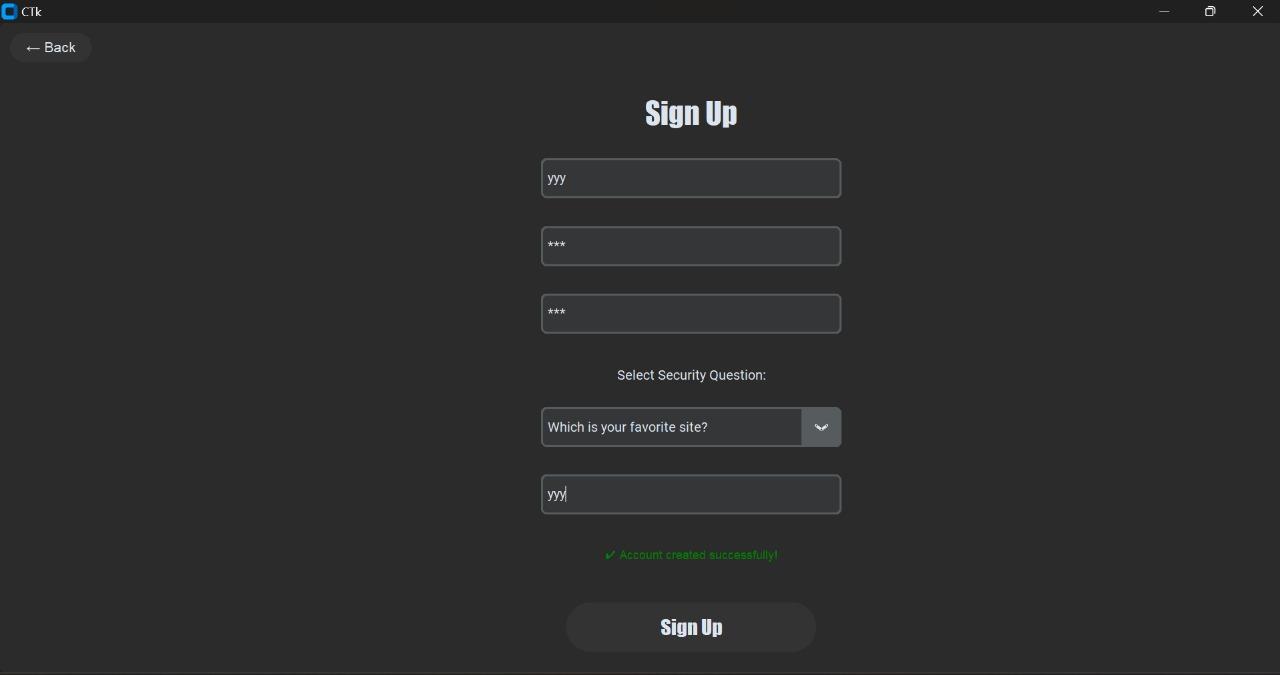
* Ensures only one session is active at a time.
* Logs out users after a period of inactivity for security.

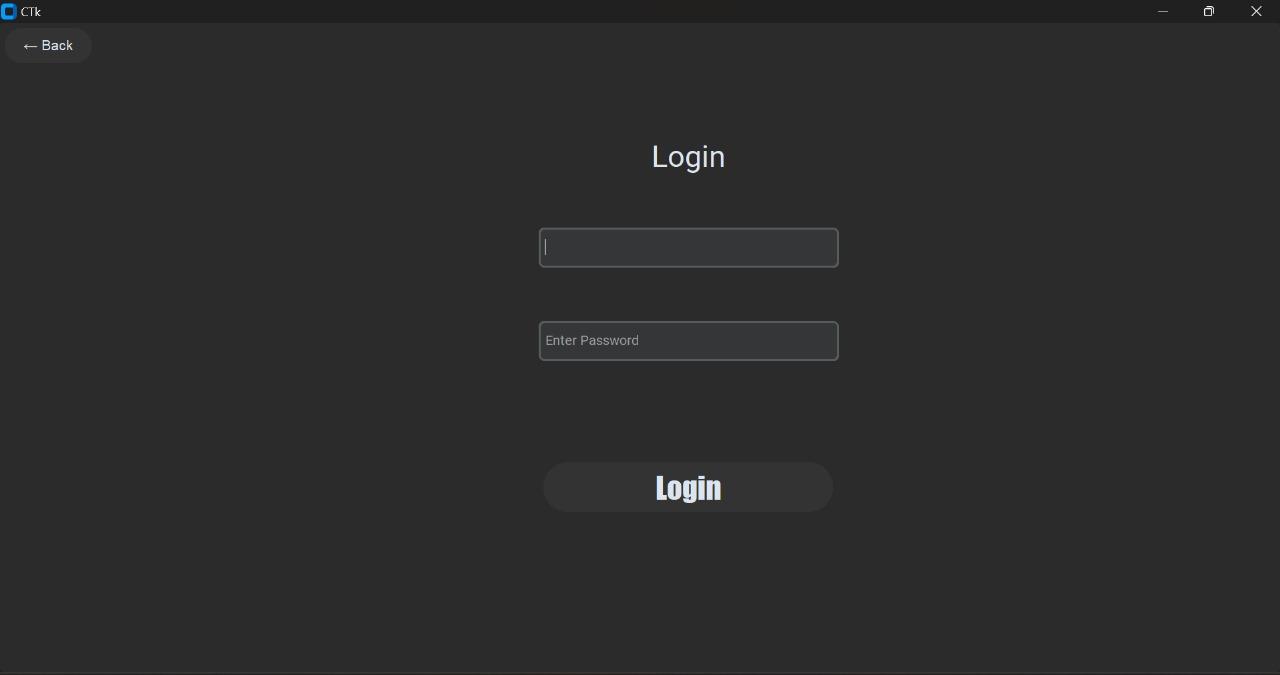
**INTERACTING** **MODULES:**

* Database Management Module: Stores user credentials securely.
* Graphical User Interface Module: Provides an interactive login/signup page.

**TECHNOLOGIES USED:**

* CustomTkinter (GUI for login/signup pages)
* SQLite/MySQL (User database)
* bcrypt (Password encryption)

 ****



**GRAPHICAL USER INTERFACE (GUI) MODULE**

**PURPOSE:**

Provides an interactive and user-friendly interface for security personnel to monitor live feeds, analyze incidents, and configure settings.

**FUNCTIONALITIES:**

**Home Page:**

* Provides navigation to key features (Live Feed, Alerts, Logs).

**Live Feed Viewer:**

* Displays real-time video surveillance with detected threats highlighted.

Incident Logs Page:

* Shows a history of detected threats and responses taken.

Settings Panel:

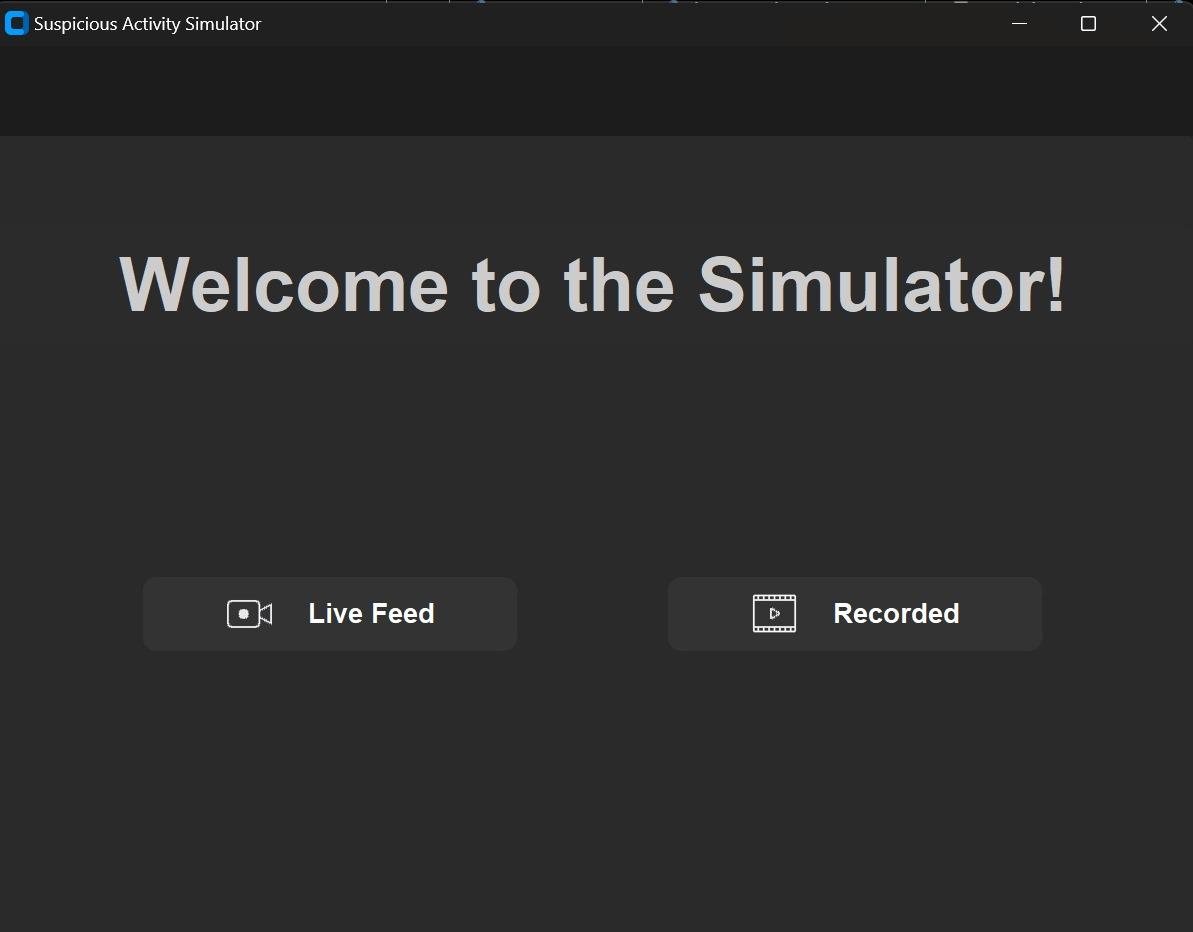
* Allows users to adjust detection sensitivity to minimize false positives.
* Lets users configure alert preferences (Email/SMS/Telegram).

**Interacting Modules:**

* All other modules (since the GUI interacts with every function of the system).

**Technologies** **Used**:

* CustomTkinter (GUI framework for Python)



**LIVE VIDEO FEED & SURVEILLANCE MODULE**

**PURPOSE:**

This module is responsible for capturing, processing, and analyzing live video feeds from cameras to detect suspicious activities such as violence, loitering, or unusual movements.

**FUNCTIONALITIES:**

**Video Capture:**

* Uses OpenCV to access webcam/CCTV feeds.
* Continuously captures frames for real-time analysis.

**Human Detection:**

* Identifies humans in the video using SSD MobileNet V2.
* Draws bounding boxes around detected individuals.

**Suspicious Activity Detection:**

* Runs frames through a violence detection model (CNN-based).
* Classifies activities as "violent" or "normal."

**Threat Level Analysis:**

* Assigns a threat score based on movement patterns and behavior.
* If the score crosses a threshold, an alert is triggered.

**Interacting Modules:**

* Automated Alert System: Sends alerts when suspicious activity is detected.
* Graphical User Interface Module: Displays real-time video feed.
* Database Management Module: Logs suspicious activity events.

**Technologies Used:**

* OpenCV (Video processing)
* TensorFlow/Keras (Deep learning model for violence detection)
* YOLOv2 (Object detection)



**RECORDED VIDEO ANALYSIS MODULE**

**PURPOSE:**

This module allows users to analyze pre-recorded videos for detecting suspicious activities, useful for reviewing past security footage.

**Functionalities:**

**Video File Selection:**

* Users select a recorded video file from their system.

**Frame Extraction & Preprocessing:**

* Splits the video into individual frames using OpenCV.
* Normalizes frames before feeding them into the model.

**Activity Classification:**

* Processes frames through the violence detection model.
* Flags frames with suspicious activity for review.

**Automated Alerts:**

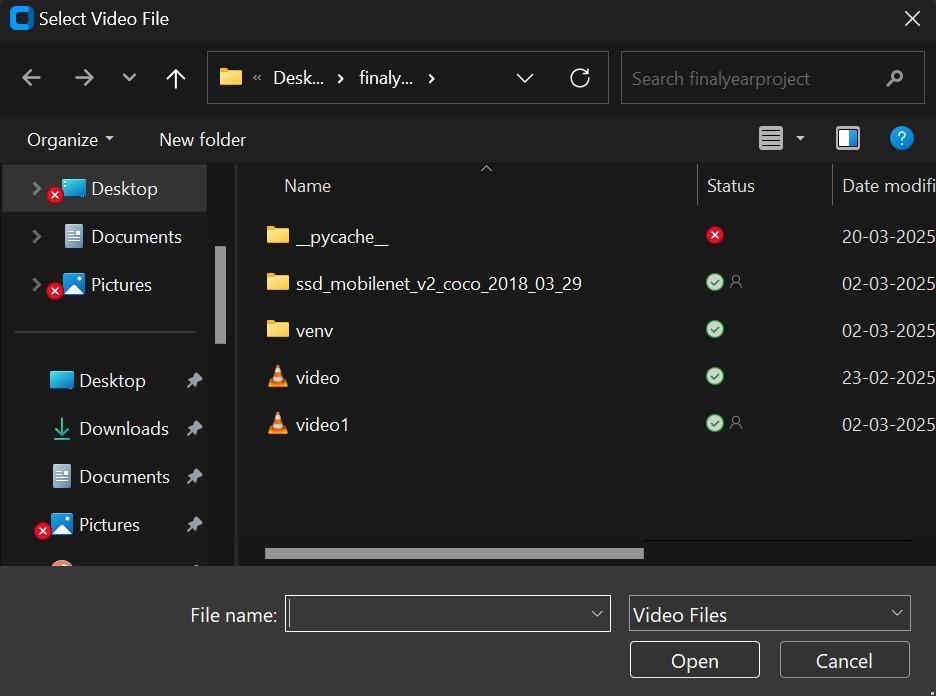
* If violence is detected, sends a Telegram notification to security personnel.

**Interacting Modules:**

* Automated Alert System: Notifies security if violence is detected.
* Graphical User Interface Module: Allows users to upload and review recorded videos.

**Technologies Used:**

* OpenCV (Video frame extraction)
* TensorFlow/Keras **(**Violence detection model)
* NumPy (Data preprocessing)





**AUTOMATED ALERT SYSTEM MODULE**

**Purpose:**

Automatically sends real-time alerts when suspicious activity is detected, ensuring quick response from security teams.

**Functionalities:**

**Real-time Notifications:**

* Sends alerts via Telegram API to the admin/security office.
* Includes timestamp, screenshot of detected activity, and location (if applicable).

**Incident Logging:**

* Stores detected incidents in a database for future reference.
* Logs details such as date, time, type of suspicious activity, and response taken.

**Admin Verification:**

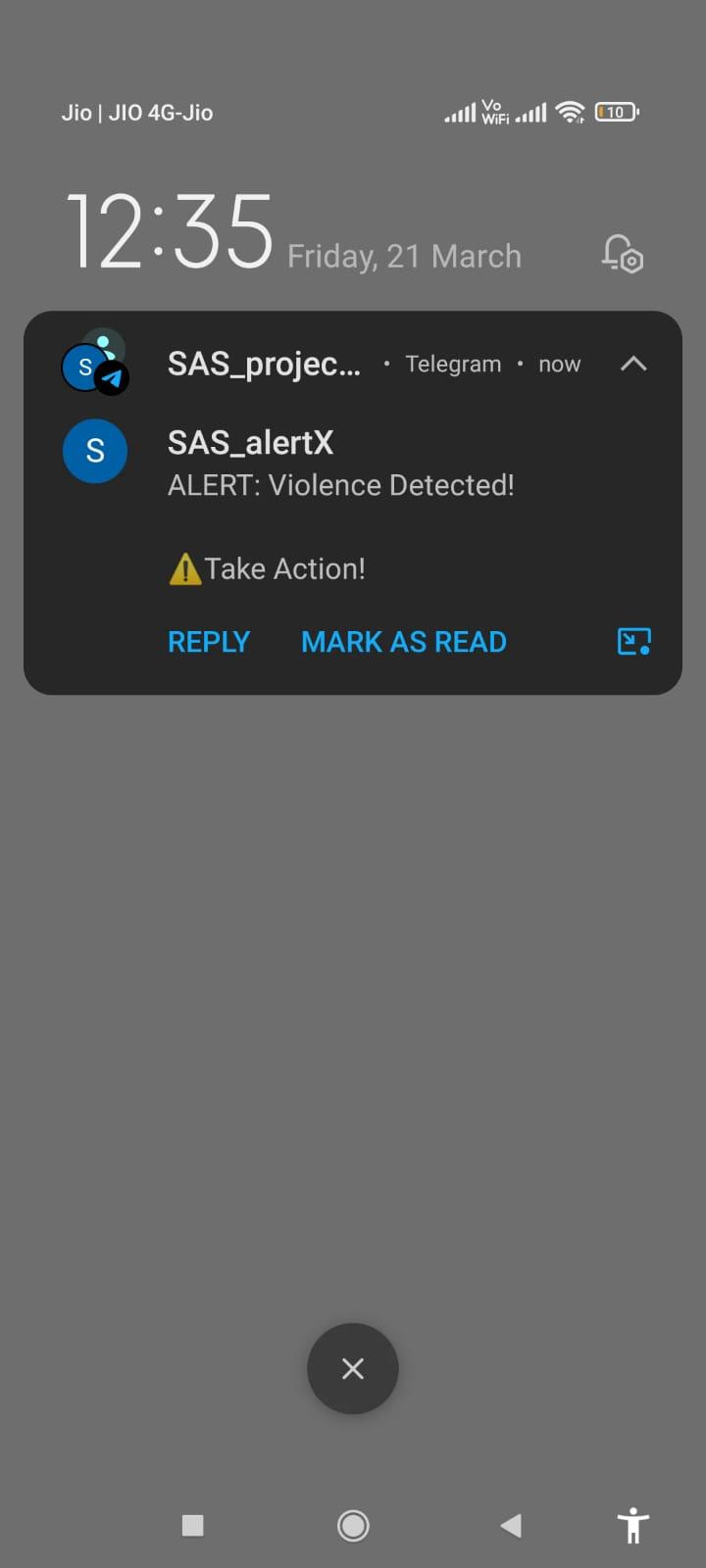
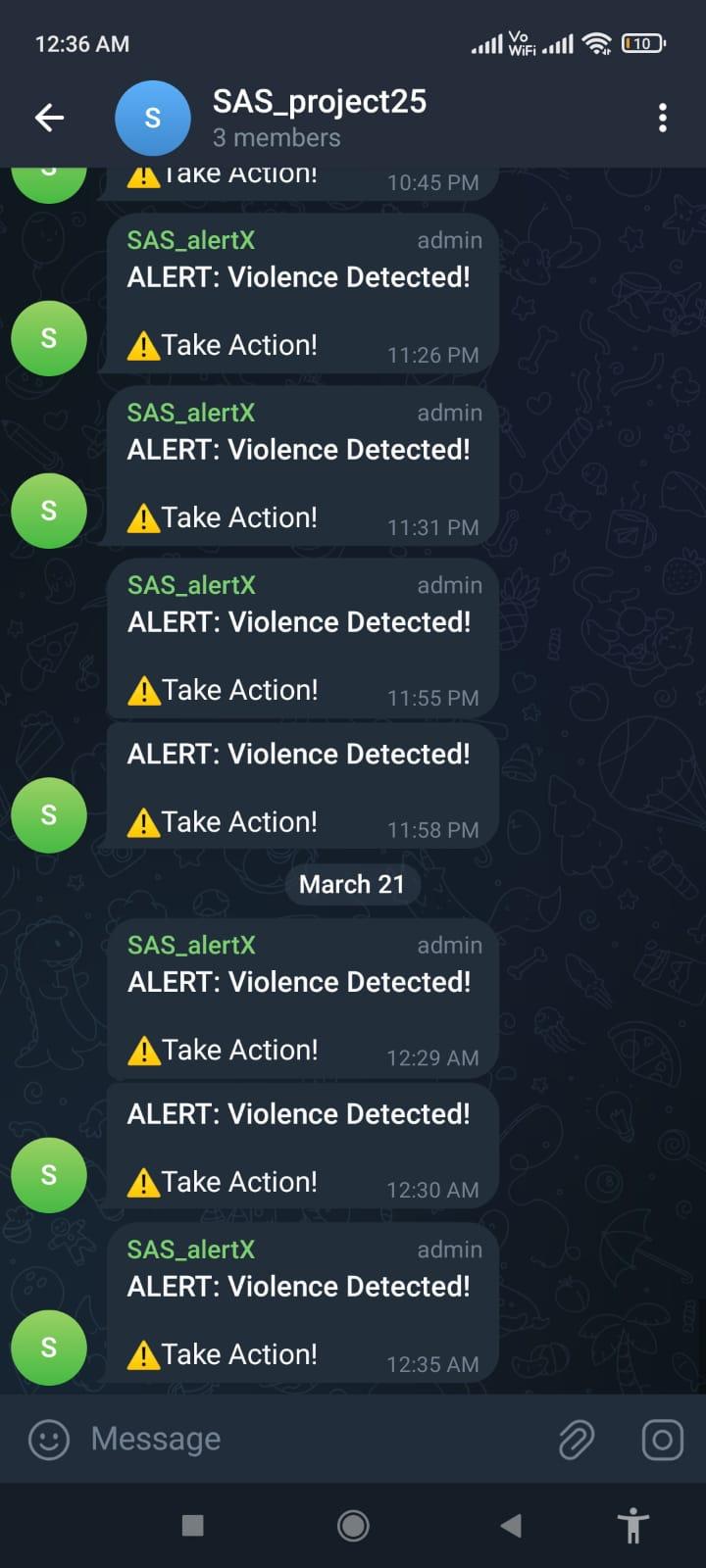
* The admin can review and confirm alerts before notifying law enforcement.

**Interacting Modules:**

* Live Video Feed & Surveillance Module: Triggers the alert when suspicious activity is detected.
* Database Management Module: Stores incident logs for future review.

**Technologies Used:**

* Telegram API (For instant alerts)
* SQLite/MySQL (For storing logs)
* Python Requests Library (For API calls)

**DATA FLOW DIAGRAMS**

**LEVEL 0 DFD (CONTEXT DIAGRAM)**

**ENTITIES & DATA FLOW:**

* User (Security Admin) → Provides video input into the system.
* Suspicious Activity Simulator →
  + Processes the video feed in real-time.
  + Detects suspicious activity (violence, unusual movement, etc.).
  + Triggers an alert notification when necessary.
  + Admin/Security Team → Receives alerts via Telegram and responds accordingly.

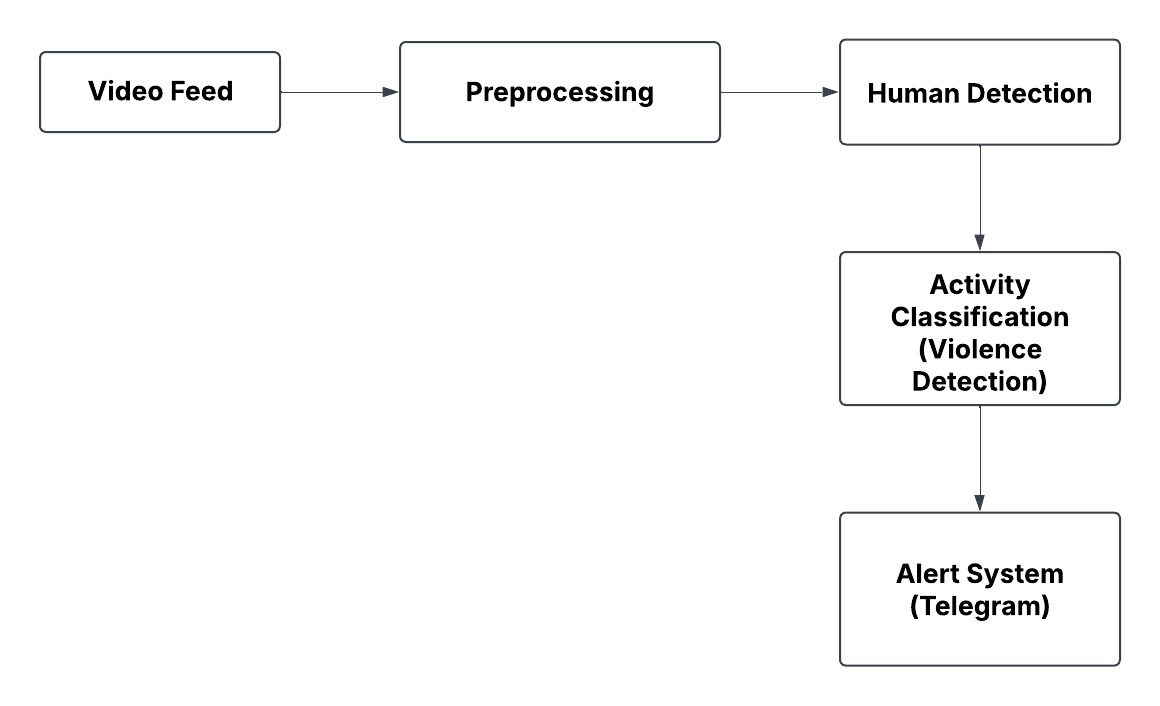
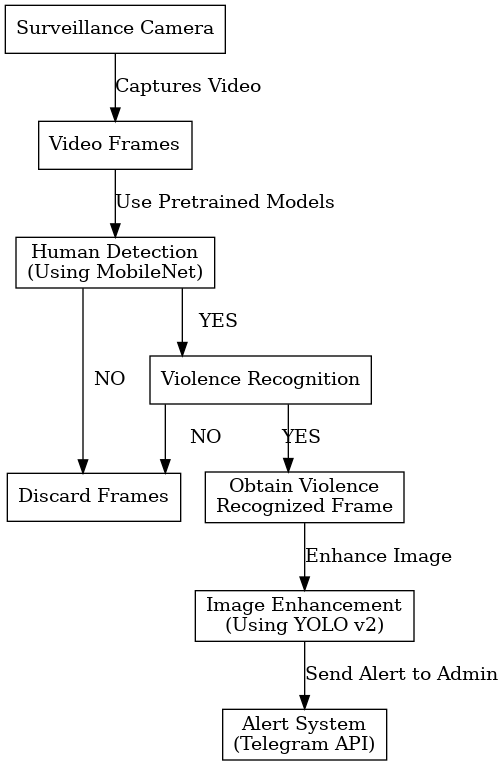


**LEVEL 1 DFD (DETAILED VIEW)**

The system mainly processes real-time video input and sends alerts.

**UPDATED SYSTEM FLOW:**

* User Provides Video Feed → Captured via OpenCV.
* Preprocessing Module → Extracts frames & prepares them for analysis.
* Human Detection Module → Identifies people in the frame using SSD MobileNet.
* Activity Classification Module → Determines if the activity is suspicious.
* Alert System → Sends real-time Telegram notifications.



**ENTITY-RELATIONSHIP (ER) DIAGRAM**

The database will have just one table.

**ER DIAGRAM STRUCTURE**

Users Table

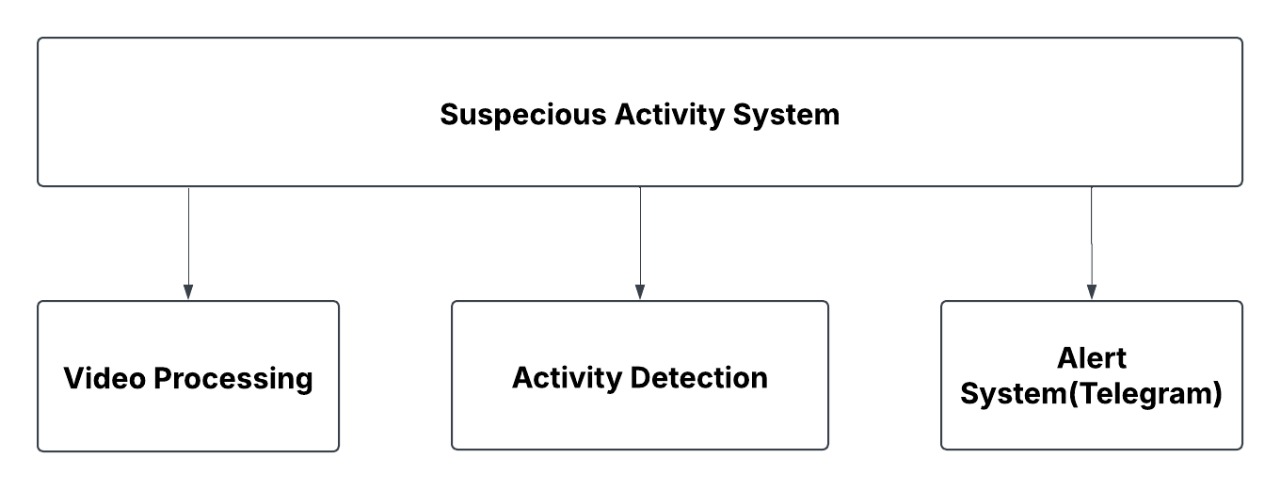
* UserID (Primary Key)
* Username
* Password (hashed)
* Security Question (hashed)

|  |
| --- |
| Users |
| UserID (PK)  Username  Password  Security Question |

**FUNCTIONAL DECOMPOSITION**

**Modules Breakdown:**

|  |  |  |
| --- | --- | --- |
| **Module** | **Description** | **Key functions** |
| user authentication | handles Login and Signup | secure login and password encryption |
| Live video processing | Detects violence in real time | Capture video, runs detection models |
| Human detection | Identifies people in the frame | SSD mobile net model |
| Activity classification | Classifies activity as Normal or violence | AI model |
| Alert system | Notifies Security team | sends telegram alerts |



**DATA DICTIONARY**

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | **Data Type** | **Description** | **Constraints** |
| UserID | INT (Primary Key) | Unique Identifier for each user | Auto Increment |
| Username | VARCHAR (255) | Stores User Login name | UNIQUE, NOT NULL |
| Password | VARCHAR (255) | Stored Encrypted Password (bcrypt hash) | NOT NULL |
| Security Question | VARCHAR (255) | Stored Encrypted answer | DEFAULT 'security' |

**INPUT DESIGN**

* Login Page → Accepts Username & Password
* Sign-Up Page → Accepts Username, Password, Security Question
* Live Video Feed → Captured automatically
* Recorded Video Upload → Users manually upload a file for analysis

**INPUTS & DATA VALIDATIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| Input Field | Module | Validation Rules | Data Types |
| Username | Login/sign-up | Must be unique, min 5 characters | VARCHAR (255) |
| Password | Login/sign-up | must be Hashed before Storing | VARCHAR (255) |
| Security Question | Sign-up | Required for password recovery | VARCHAR (255) |
| Video Feed | Video Processing | Captured automatically | OpenCV Video Streams |
| Recorded Video | Video Analysis | MP4, AVI formats allowed | File Uploads |

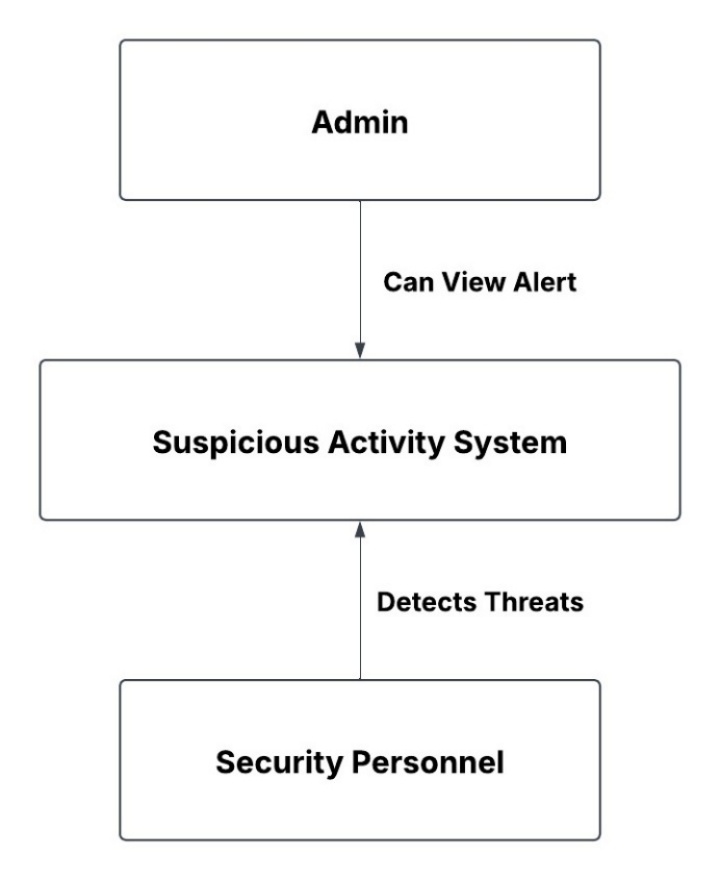
**UML / USE CASE DIAGRAM**

**Actors in the System:**

* Admin → Manages users, receives alerts.
* Security Personnel → Monitors live feed, receives alerts.
* Suspicious Activity System → Processes video, detects threats, sends alerts.

**Use Cases:**

* User Login → Admin/Security logs in to access the system.
* Sign-Up → New users register with credentials.
* Live Video Processing → System continuously scans video feed.
* Recorded Video Analysis → User uploads a video for analysis.
* Threat Detection → AI classifies activity as suspicious or normal.
* Send Alert → System notifies security via Telegram.



**DATABASE** **DESIGN**

**Purpose:**

The database is designed only to store user authentication details.

Users Table → Stores login credentials and user roles.

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Constraints |
| UserID | INT (Primary Key) | Unique Identifier for each user | Auto Increment |
| Username | VARCHAR (255) | Stores User Login name | UNIQUE, NOT NULL |
| Password | VARCHAR (255) | Stored Encrypted Password (bcrypt hash) | NOT NULL |
| Security Question | VARCHAR (255) | Stored Encrypted Security Answer | DEFAULT 'security' |

**DATA STRUCTURE**

The project only deals with user credentials, the data structure is straightforward.

**How Data is Stored in the Users Table**

* When a new user signs up → The username & password are stored in a hashed format using bcrypt.
* When a user logs in → The system retrieves the hashed password and verifies it.

****

{ "UserID":"project25",

"Username": "AdminUser",

"Password": "$2b$12$hGfh8jX.TyR7Xj9pXkYhL.",

"Security Question": " e99a18c428cb38d5f260853678922e03 "

}

**CODING**

**Tools & Technologies Used**

The Suspicious Activity Simulator was built using a combination of machine learning, computer vision, and GUI frameworks to enable real-time detection of suspicious activities. Below are the key tools and technologies used in the project:

|  |  |
| --- | --- |
| Tool/Technology | Purpose |
| Python | Core Programming Language for Development |
| OpenCV | Video processing and real-time frame analysis |
| TensorFlow | AI model for violence detection |
| Custom Tkinter | GUI framework for User interface |
| SQLite | Database for storing user credentials |
| bcrypt | Secure password hashing for authentication |
| Telegram API | Sending real time alerts to security Personnel |

**WHY PYTHON FOR DEVELOPMENT?**

Python was chosen as the primary programming language for this project because of its extensive libraries, ease of integration, and strong support for AI and computer vision tasks. The key reasons include:

* Rich AI & ML Ecosystem → Python provides libraries like TensorFlow, Keras, and OpenCV, which are essential for training and deploying machine learning models.
* Flexibility & Rapid Development → Python's simple syntax allows for faster prototyping and development compared to other languages like C++ or Java.
* Strong Support for Computer Vision → OpenCV, a Python-compatible library, is widely used for image and video processing, making real-time detection easier.
* Cross-Platform Compatibility → Python runs on Windows, Linux, and macOS, ensuring that the system can be deployed in various environments.
* Community & Documentation → Python has one of the largest developer communities, ensuring strong support and extensive documentation for troubleshooting and improvements.

**DATA VALIDATIONS**

**Purpose of Data Validation:**

Data validation ensures that only correct and secure input is processed by the system.

|  |  |  |
| --- | --- | --- |
| Module | Validation Rule | Key Functions |
| Username | Min 5 characters, no special symbols | Valid: securityAdmin / invalid:Admin |
| password | Min 8 characters, must include a number & symbols | Valid: Pass123 / Invalid: password |
| Recorded Video | only MP4/AVI formats allowed | Valid: security\_footage.mp4 |
| Live Feed Processing | frame rate should be stable (FPS>15) | Valid:30FPS / Invalid: 5 FPS |

**SAMPLE CODE SNIPPETS**

**USER AUTHENTICATION (SIGN-UP & LOGIN**)

import sqlite3

import bcrypt

import customtkinter as ctk

# Database Connection

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS users (UserID INTEGER PRIMARY KEY, Username TEXT UNIQUE, Password TEXT, Role TEXT)")

conn.commit()

# Hashing Password

def hash\_password(password):

return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()

# Sign-Up Function

def signup(username, password, role="Security"):

hashed\_pwd = hash\_password(password)

try:

cursor.execute("INSERT INTO users (Username, Password, Role) VALUES (?, ?, ?)", (username, hashed\_pwd, role))

conn.commit()

print(" User Registered Successfully!")

except sqlite3.IntegrityError:

print("Username already exists!")

# Login Function

def login(username, password):

cursor.execute("SELECT Password FROM users WHERE Username=?", (username,))

user = cursor.fetchone()

if user and bcrypt.checkpw(password.encode(), user[0].encode()):

print("Login successful!")

else:

print("Invalid username or password!")

# Example Usage

signup("AdminUser", "Pass@123")

login("AdminUser", "Pass@123")

**HOW PASSWORD HASHING PREVENTS DATA BREACHES**

Password hashing is a critical security measure that protects user credentials from being exposed in case of a data breach. Instead of storing plaintext passwords, our system hashes them using bcrypt, making it nearly impossible for attackers to retrieve the original password.

**WHAT IS PASSWORD HASHING?**

Hashing is a one-way cryptographic function that converts a password into an irreversible, fixed-length string. Unlike encryption, hashing cannot be reversed to retrieve the original password.

**Example:**

* User Password: MySecurePass123
* Hashed Output: $2b$12$T3NvZk1...YXdNeYTY3X

**Every time a user logs in, the system:**

* Hashes the entered password and compares it to the stored hash.
* If the two hashes match, access is granted.
* Even if an attacker steals the database, they only get hashed passwords—not actual credentials.

**Why is Hashing Important for Security?**

|  |  |
| --- | --- |
| Security Threat | How Hashing prevents it |
| Data Breaches | Even if hackers steal the database, they only get hashed passwords, which are useless without decryption |
| Burt force Attacks | Hashing algorithms are computationally expensive, making it impractical for attackers to guess the passwords |
| Rainbow Table Attack’s | Hashing uses salting, adding a unique random value to each password, preventing attackers from using precomputed hash lookups |

**WHY WE USE BCRYPT FOR PASSWORD HASHING**

Python’s bcrypt library is used in our system for hashing passwords because:

* It generates a unique salt for each password, making hash values different even for identical passwords.
* It is computationally expensive, slowing down brute force attacks.
* It automatically adapts to future hardware, ensuring long-term security.

**PASSWORD HASHING IN PYTHON**

import bcrypt

# Hashing a password before storing in the database

def hash\_password(password):

salt = bcrypt.gensalt() # Generate a random salt

hashed\_password = bcrypt.hashpw(password.encode(), salt)

return hashed\_password.decode()

# Verifying a password during login

def check\_password(stored\_hash, entered\_password):

return bcrypt.checkpw(entered\_password.encode(), stored\_hash.encode())

# Example Usage

user\_password = "SecurePass123"

hashed = hash\_password(user\_password)

print("Hashed Password:", hashed)

# Simulate login attempt

print("Password Match:", check\_password(hashed, "SecurePass123")) # True

print("Wrong Password Match:", check\_password(hashed, "WrongPass")) # False

**LIVE VIDEO PROCESSING (USING OPENCV)**

import cv2

# Open the webcam

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

if not ret:

break

cv2.imshow("Live Video Feed", frame)

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

break

cap.release()

cv2.destroyAllWindows()

**USING OPENCV FOR LIVE VIDEO CAPTURE & AI-BASED FRAME PROCESSING**

The Suspicious Activity Simulator utilizes OpenCV for real-time video processing. OpenCV enables the system to capture live video feeds from a webcam or CCTV camera and process each frame individually for AI-based violence detection.

**How OpenCV Captures Live Video Feeds**

OpenCV (Open-Source Computer Vision Library) is a widely used library for image and video processing. In this project, it is used to:

* Access a live camera feed from a webcam or an external CCTV.
* Capture and display video frames continuously.
* Pass each frame to the AI model for suspicious activity detection.

**STEPS IN LIVE VIDEO CAPTURE:**

* Open the camera using cv2.VideoCapture(0).
* Read video frames continuously using cap.read().
* Display the video stream in a GUI window.
* Process each frame before passing it to the AI model for classification.

**CODE TO CAPTURE LIVE VIDEO USING OPENCV:**

import cv2

# Open the webcam (0 for default camera)

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read() # Read each frame from the webcam

if not ret:

break # Stop if the camera is not accessible

cv2.imshow("Live Video Feed", frame) # Display the video feed

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

break

cap.release()

cv2.destroyAllWindows()

**Frame-by-Frame Processing for AI-Based Violence Detection**

**AI models process images (not video files directly), the system must:**

* Extract each frame from the live video feed.
* Resize and normalize the frame to match the AI model's input shape.
* Convert the frame into a format suitable for deep learning models.
* Classify the frame as "Violent" or "Normal" using the trained AI model.

**Steps in AI-Based Frame Processing:**

* Capture a frame from the live video using OpenCV.
* Preprocess the frame (resize, normalize, convert to array).
* Feed the processed frame into the trained AI model.
* Receive a classification label ("Violent" or "Normal").
* Trigger an alert if violence is detected.

**Code to Process Frames for AI-Based Detection:**

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

# Load Pretrained AI Model

model = load\_model("violence\_detection\_model.h5")

cap = cv2.VideoCapture(0) # Open webcam

while True:

ret, frame = cap.read() # Capture each frame

if not ret:

break

# Preprocess Frame for AI Model

frame\_resized = cv2.resize(frame, (64, 64)) # Resize to model input size

frame\_normalized = frame\_resized / 255.0 # Normalize pixel values

frame\_input = np.expand\_dims(frame\_normalized, axis=0) # Expand dimensions for model input

# AI Model Prediction

prediction = model.predict(frame\_input)

label = "Violent" if prediction[0] > 0.5 else "Normal"

# Display Classification Result

cv2.putText(frame, f"Activity: {label}", (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

cv2.imshow("AI-Based Activity Detection", frame)

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

break

cap.release()

cv2.destroyAllWindows()

**VIOLENCE DETECTION USING TENSORFLOW/KERAS**

from tensorflow.keras.models import load\_model

import numpy as np

# Load Pretrained Model

model = load\_model("violence\_detection\_model.h5")

def predict\_violence(frame):

frame\_resized = cv2.resize(frame, (64, 64)) # Resize for model

frame\_normalized = frame\_resized / 255.0

frame\_input = np.expand\_dims(frame\_normalized, axis=0)

prediction = model.predict(frame\_input)

return "Violent" if prediction[0] > 0.5 else "Normal"

# Example Usage

frame = cv2.imread("sample\_frame.jpg")

print(predict\_violence(frame))

**AI Model Classification: How Frames Are Classified as Violent or Normal**

The Suspicious Activity Simulator, the AI model processes each frame from a live video feed and classifies it as either "Violent" or "Normal". The classification process involves image preprocessing, feeding the frame into the model, and obtaining a prediction.

**HOW THE AI MODEL CLASSIFIES FRAMES**

Step 1: Capture a frame from the live video feed.  
Step 2: Preprocess the frame (resize, normalize, format conversion).  
Step 3: Feed the frame into the trained AI model.  
Step 4: AI model predicts the probability of violence.  
Step 5: If probability > threshold (e.g., 0.5), classify as "Violent"; otherwise, classify as "Normal".

**Image Preprocessing Steps Before Prediction**

Before the AI model can classify an image, the captured video frame must be preprocessed. The following preprocessing steps ensure that the model can interpret the frame correctly:

**Resize the Image**

* The AI model expects images of a fixed size (e.g., 64x64 pixels).
* Resizing ensures that all input frames have consistent dimensions.

frame\_resized = cv2.resize(frame, (64, 64))

Normalize Pixel Values

* Deep learning models work best with normalized inputs.
* Pixel values (0 to 255) are converted to a 0-1 range for better model efficiency.

frame\_normalized = frame\_resized / 255.0 # Normalize pixel values

**Convert Frame into a Model-Compatible Format**

* Models expect inputs in the format of (batch\_size, width, height, channels).
* Expand dimensions to add batch size = 1 before feeding into the model.

frame\_input = np.expand\_dims(frame\_normalized, axis=0) # Add batch dimension

**Human Detection Code Snippet**

Human Detection Module (Using SSD MobileNet & OpenCV)

The system detects humans in the video feed using the SSD MobileNet model, which is optimized for object detection.

import cv2

# Load Pre-trained Model

model\_path = "ssd\_mobilenet\_v2.pb"

net = cv2.dnn.readNetFromTensorflow(model\_path)

# Capture Video

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

if not ret:

break

# Convert Frame to Blob

blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (127.5, 127.5, 127.5), swapRB=True, crop=False)

net.setInput(blob)

detections = net.forward()

# Draw Bounding Box for Detected Humans

for detection in detections[0, 0, :, :]:

confidence = detection[2]

if confidence > 0.5: # Confidence threshold

print("Human detected!")

cv2.imshow("Human Detection", frame)

if cv2.waitKey(1) & 0xFF == ord("q"):

break

cap.release()

cv2.destroyAllWindows()

**SSD MobileNet & COCO Dataset for Human Detection**

**What is SSD MobileNet?**

SSD MobileNet (Single Shot MultiBox Detector with MobileNet) is a lightweight deep-learning model used for real-time object detection. It is specifically designed to:

* Detect humans and objects in video frames with high speed and efficiency.
* Run on low-power devices like Raspberry Pi, edge devices, or CPUs.
* Provide fast inference time for real-time applications like surveillance.

**Why Use SSD MobileNet for Human Detection?**

* Unlike YOLO (You Only Look Once), which is more complex, SSD MobileNet is optimized for fast and efficient detection**.**
* It balances speed and accuracy, making it ideal for live video processing**.**
* The model is pre-trained on the COCO dataset, which includes human detection classes.

**COCO Dataset for Object Detection**

**What is COCO?**

COCO (Common Objects in Context) is a large-scale dataset used for object detection, segmentation, and classification. It contains:

* 80 object categories (e.g., person, vehicle, animals, objects).
* Over 200,000 labeled images for training AI models.
* Annotations for detecting objects in various conditions (day/night, occlusions, etc.).

**Why Use COCO for Human Detection?**

* COCO has thousands of images of people in different environments, making it ideal for training human detection models.
* Since SSD MobileNet is pre-trained on COCO, it can detect humans with high accuracy without additional training.

**How SSD MobileNet Detects Humans in Video**

1️ Load the SSD MobileNet model (pre-trained on COCO).  
2️Capture a frame from the live video using OpenCV.  
3️ Convert the frame into a blob (preprocessing for the model).  
4️ Run the blob through SSD MobileNet for object detection.  
5️ Check if a "person" class is detected in the frame.  
6️ Draw bounding boxes around detected humans**.**

**Violence Detection Code Snippet**

Violence Detection Module (Using CNN Model)

The AI model classifies video frames as “Violent” or “Normal” using a Convolutional Neural Network (CNN) trained on an action recognition dataset.

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

# Load Trained CNN Model

model = load\_model("violence\_detection\_model.h5")

def predict\_violence(frame):

# Preprocess Frame

frame\_resized = cv2.resize(frame, (64, 64))

frame\_normalized = frame\_resized / 255.0

frame\_input = np.expand\_dims(frame\_normalized, axis=0)

# Predict

prediction = model.predict(frame\_input)

return "Violent" if prediction[0] > 0.5 else "Normal"

# Example Usage

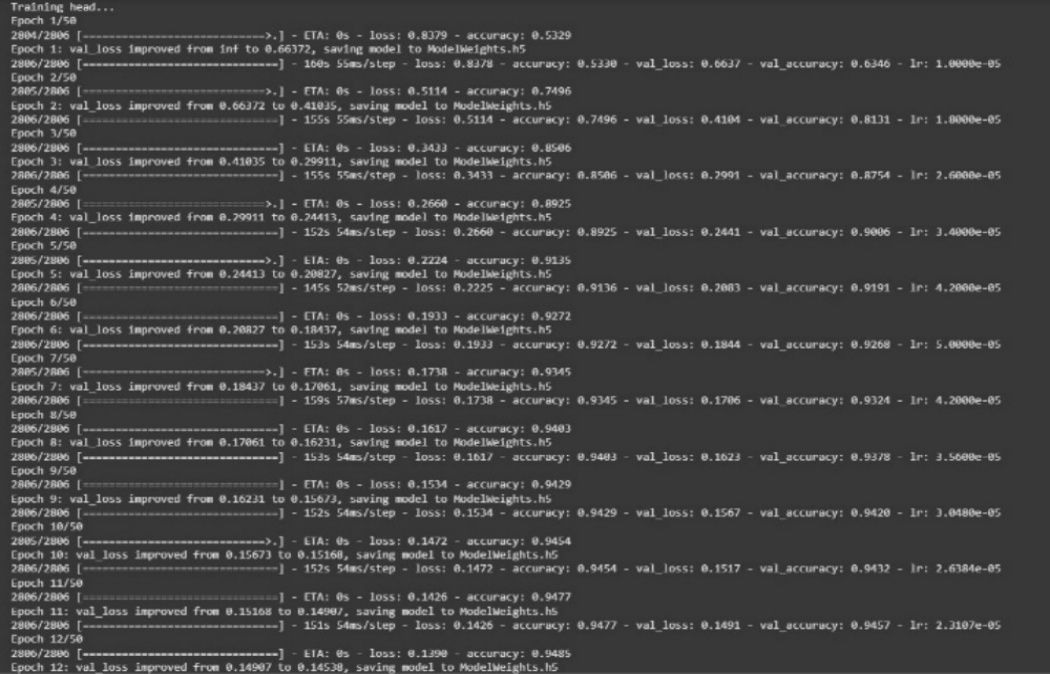
frame = cv2.imread("test\_frame.jpg")

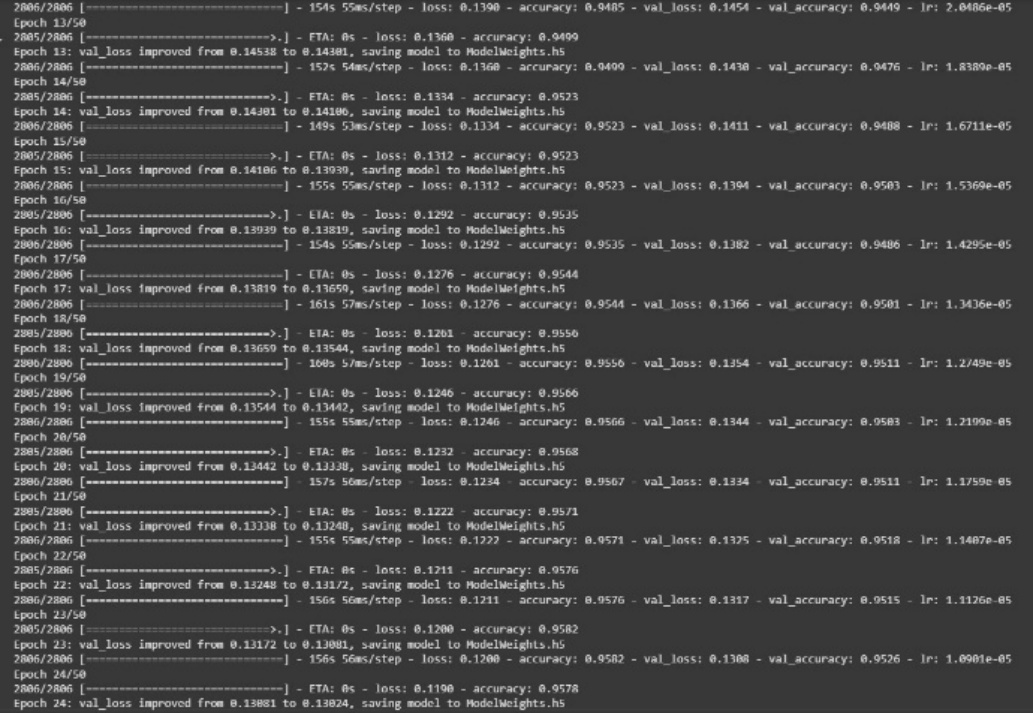
print("Prediction:", predict\_violence(frame))

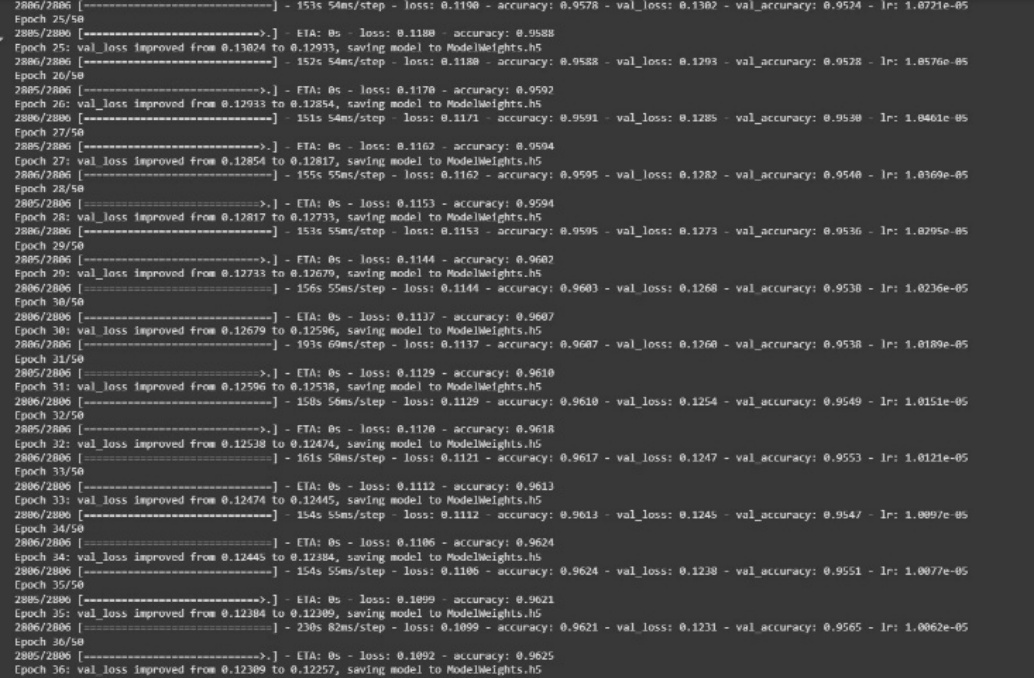
Training Results of the AI Model

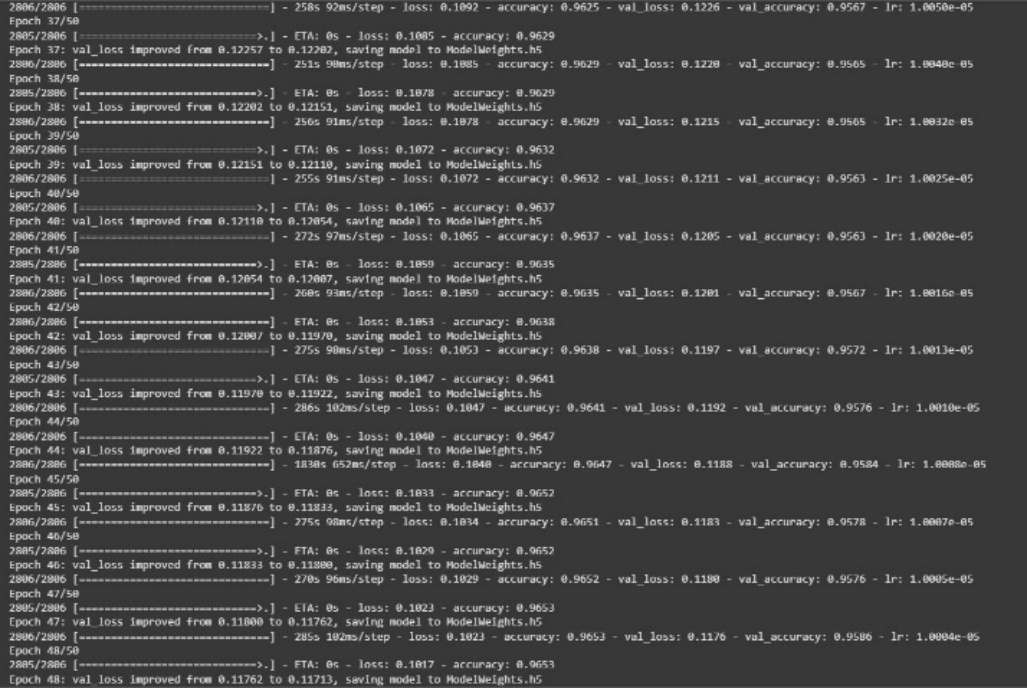
The violence detection model was trained using a dataset of action recognition videos. Below is the model’s accuracy and loss over epochs:

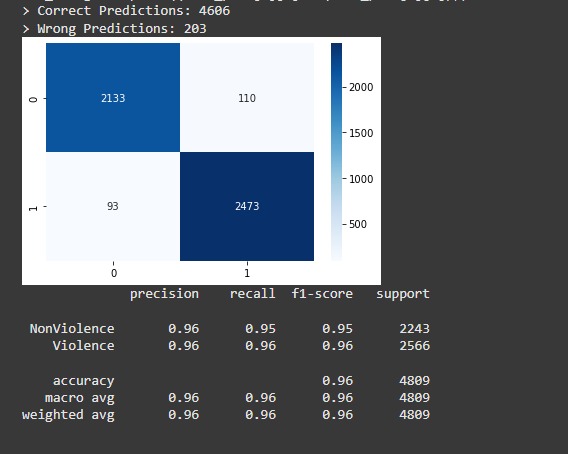


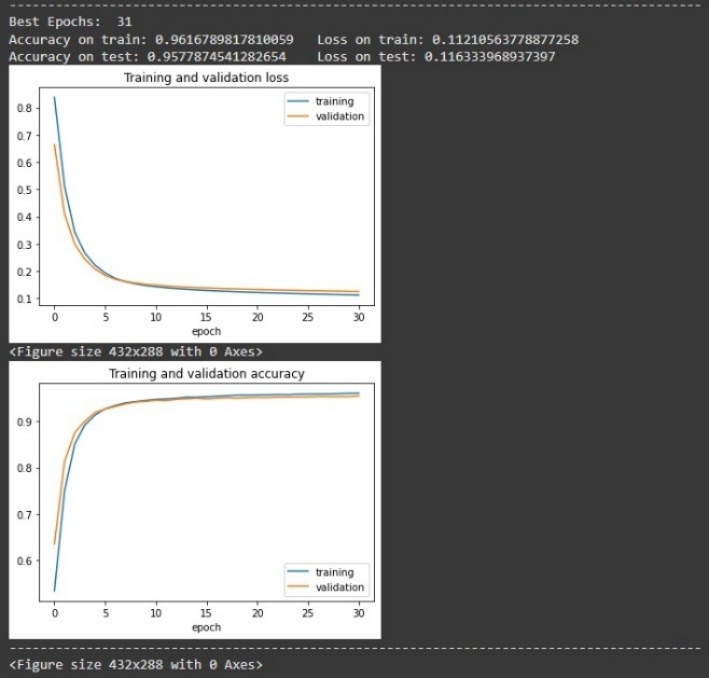












Final Model Performance

|  |  |
| --- | --- |
| Metric | Value |
| Training Accuracy | 96.50% |
| Validation Accuracy | 88.30% |
| False Alarm Rate | 7.20% |
| Precision (violence Detection) | 89.70% |

**AI Model Prediction**

Once preprocessing is complete, the processed frame is passed to the AI model to determine if the activity is violent or normal.

**Prediction Process:**

* Frame is fed into the trained deep learning model (e.g., CNN).
* Model outputs a probability score (e.g., 0.85 for "Violent").
* Compare the probability against a threshold (e.g., 0.5).
* If probability > 0.5, classify as "Violent"; otherwise, classify as "Normal".

**AI Model Classification Code**

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

# Load Trained AI Model

model = load\_model("violence\_detection\_model.h5")

def predict\_violence(frame):

# Preprocessing Steps

frame\_resized = cv2.resize(frame, (64, 64)) # Resize to model input size

frame\_normalized = frame\_resized / 255.0 # Normalize pixel values

frame\_input = np.expand\_dims(frame\_normalized, axis=0) # Expand dimensions for model

# Get AI Model Prediction

prediction = model.predict(frame\_input)

probability = prediction[0][0] # Probability score

# Classify Based on Probability Threshold

return "Violent" if probability > 0.5 else "Normal"

# Example Usage

frame = cv2.imread("test\_frame.jpg") # Load a test frame

classification = predict\_violence(frame)

print("AI Classification:", classification) # Output: "Violent" or "Normal"

Sending Alerts via Telegram API

import requests

TELEGRAM\_BOT\_TOKEN = "your\_bot\_token"

CHAT\_ID = "your\_chat\_id"

def send\_alert(message):

url = f"https://api.telegram.org/bot{TELEGRAM\_BOT\_TOKEN}/sendMessage"

payload = {"chat\_id": CHAT\_ID, "text": message}

requests.post(url, data=payload)

# Example Usage

send\_alert("Suspicious activity detected!")

**Instant Alerts via Telegram API**

The Suspicious Activity Simulator, the system sends real-time alerts via the Telegram API whenever suspicious activity (violence) is detected. This ensures that security personnel are notified immediately so they can take action.

**How Alerts Are Sent Instantly via Telegram API**

* Step 1: AI model detects violence in a video frame.
* Step 2: The system triggers the send\_alert() function.
* Step 3: A notification is sent to a predefined Telegram chat or group.
* Step 4: Security personnel receive the alert instantly, even if they are not near the monitoring system.

**Code to Send Alerts via Telegram API**

import requests

import os

# Securely load API keys from environment variables

TELEGRAM\_BOT\_TOKEN = os.getenv("TELEGRAM\_BOT\_TOKEN")

CHAT\_ID = os.getenv("TELEGRAM\_CHAT\_ID")

def send\_alert(message):

"""Sends an instant alert message to a Telegram group or user."""

url = f"https://api.telegram.org/bot{TELEGRAM\_BOT\_TOKEN}/sendMessage"

payload = {"chat\_id": CHAT\_ID, "text": message}

response = requests.post(url, data=payload)

if response.status\_code == 200:

print("Alert Sent Successfully!")

else:

print(f"Alert Failed! Error Code: {response.status\_code}")

# Example Usage

send\_alert("ALERT: Suspicious activity detected in the surveillance area!")

**Implementation**

The Implementation section in your documentation explains how the project is set up, integrated, and deployed. It should cover:

System Setup & Installation

Steps to Install & Run the System:

Install Dependencies  
 Run the following command to install required Python libraries:

pip install opencv-python tensorflow customtkinter bcrypt sqlite3 requests

Set Up the Database (User Authentication Only)

import sqlite3

conn = sqlite3.connect("users.db")

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

UserID INTEGER PRIMARY KEY AUTOINCREMENT,

Username TEXT UNIQUE NOT NULL,

Password TEXT NOT NULL,

Role TEXT DEFAULT 'Security'

)

""")

conn.commit()

conn.close()

Run the Application

* Open VS Code or PyCharm.
* Run the main Python script to start the GUI and detection system.

python main.py

**Module Integration**

**How Different Modules Work Together**

|  |  |
| --- | --- |
| Module | Integration |
| User Authentication | Linked to GUI(Login/sign-up) and SQLite DB |
| Live Video Processing | Captures frames passes to AI model for analysis |
| AI-Based Activity Detection | Uses Tensor Flow model to classify video frames |
| Alert System | Sends alerts via Telegram API when suspicious activity is detected |

**Deployment & Execution**

**Deployment Options:**

* Local Deployment → Run on a PC with a webcam (ideal for testing).
* Server Deployment → Can be hosted on a Raspberry Pi or cloud server for real-time monitoring.

**Challenges Faced & Solutions**

|  |  |
| --- | --- |
| **Challenges** | **Solution Implemented** |
| **Low FPS in Live Feed** | Optimized OpenCV frame capture settings. |
| **False Positives in Detection** | Adjusted AI model thresholds for better accuracy. |
| **Password Security Risk** | Used bcrypt hashing to store passwords securely. |

**Planning & Scheduling**

The timeline of development, including task breakdown, milestones, and deadlines for your Suspicious Activity Simulator project.

**Project Timeline & Phases**

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Task | Start Date | End Dae |
| Phase 1 | Project proposal and Research | 13-01-2025 | 21-01-2025 |
| Phase 2 | System Design (DFD, ERF, UML) | 16-01-2025 | 25-01-2025 |
| Phase 3 | Database setup (User Authentication) | 24-01-2025 | 27-01-2025 |
| Phase 4 | GUI Development (Login & sign-up) | 28-01-2025 | 4-02-2025 |
| Phase 5 | Video Processing (Live Feed) | 16-02-2025 | 28-02-205 |
| Phase 6 | AI Model Integration (Violence detection | 22-02-2025 | 28-02-2025 |
| Phase7 | Alert System (Telegram API) | 3-03-2025 | 7-03-2025 |
| Phase 8 | Testin and Debugging | 10-03-2025 | 19-03-2025 |
| Phase 9 | Final Documentation& Report Submission | 14-03-2025 | 21-03-2025 |

**Weekly Task Breakdown**

|  |  |  |
| --- | --- | --- |
| week | Task to complete | Team Member Responsibility |
| week1 | System Design, DFD, ER Diagram | Aryan Kujur |
| week2 | Database &GUI Implementation | Dishant Uprety and Nihal Shukla |
| week3 | Live Video Processing | Md Muzakkir Hussain |
| week4 | AI Model Training& Testing | Md Muzakkir Hussain and Dishant Uprety |
| week5 | Alert System and Integration | Dishant Uprety |
| week6 | Testing and Debugging | Nihal Shukla |
| week7 | Final Report &Documentation | Md Muzakkir Hussain and Aryan Kujur |

**TESTING**

The testing process, including test strategies, test cases, expected results, and roles & responsibilities in ensuring the system works as intended.

**Test Strategies**

**The following testing strategies were used to validate the system:**

|  |  |
| --- | --- |
| **Test Strategy** | **Description** |
| |  | | --- | | Unit Testing |  |  | | --- | |  | | |  | | --- | | Individual modules such as Login, Video Processing, AI Model, and Alerts were tested separately. |  |  | | --- | |  | |
| |  | | --- | | Integration Testing |  |  | | --- | |  | | |  | | --- | | Ensured that all modules interact correctly, such as AI processing video frames and triggering alerts. |  |  | | --- | |  | |
| |  | | --- | | Functional Testing |  |  | | --- | |  | | |  | | --- | | Verified that the login system, AI model, and alert system work according to requirements. |  |  | | --- | |  | |
| |  | | --- | | Performance Testing |  |  | | --- | |  | | |  | | --- | | Tested the real-time processing speed of video feeds and response time of the AI model. |  |  | | --- | |  | |
| |  | | --- | | Security Testing |  |  | | --- | |  | | Checked passwordencryption and ensured unauthorizeduserscannot access the system**.** |

Test Cases

Each test case includes:

* Test Class ID → Unique identifier for the test.
* Test Condition → What is being tested?
* Expected Result → What should happen if the system is correct?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Test Conditions | Expected results | Actual Result | Status |
| TC001 | Login with Valid credentials | User logs in successfully | User Logged in | Pass |
| TC002 | Login with incorrect password | "Incorrect Password" error displayed | "Incorrect Password" Displayed | Pass |
| TC003 | Password stored securely | Password is stored as Hashed value(bcrypt) | Password encrypted correctly | Pass |
| TC004 | webcam connectivity check | Video feed Starts without lag | video feed opened successfully | Pass |
| TC005 | AI Model Detects violence correctly | Alert sent if violence is detected | Alert sent on violent activity | Pass |
| TC006 | False alarm prevention | Normal Behavior should not trigger an alert | No Alert triggered for normal activity | False Positive Detected |
| TC007 | Alert system sends notification | Telegram Alert received in real Time | Alert received on Telegram | Pass |

* Actual Result → What actually happened?
* Pass/Fail Status → Did the test succeed?

**Test Class ID & Test Conditions**

|  |  |
| --- | --- |
| Test Class ID | Test Condition |
| TC001 | User Login with valid credentials |
| TC002 | User Login with incorrect Password |
| TC003 | Secure password storage using bcrypt |
| TC004 | Webcam successfully Captures video feed |
| TC005 | AI correctly classifies violent behaviors |
| TC006 | False Positive Rate in AI detection |
| TC007 | Telegram Alert System sends notification |

**Roles & Responsibilities**

|  |  |  |
| --- | --- | --- |
| Role | Responsibility | Assigned To |
| Retest Engineer | writing test cases & conducting manual tests | Aryan Kujur |
| Developer | Fixing Bugs found during tests | Nihal Shukla |
| AI Model Tester | Verifying accuracy of AI predictions | Md Muzakkir Hussain |
| Security Analyst | Checking login & password encryption security | Dishant Uprety |

**Future Enhancements**

Although the current system successfully detects suspicious activities, there are several areas for improvement.

|  |  |
| --- | --- |
| Enhancement | Description |
| Improved AI model | Enhance the model with more diverse datasets to improve accuracy. |
| Multi-camera Support | Allow Monitoring from multiple CCTV camera simultaneously. |
| Cloud Storage Integration | Option to store suspicious frames on a cloud server for later review. |
| Facial Recognition | Identify known suspects using face recognition AI models. |
| Voice and Sound Detection | Analyze audio data to detect gunshots, screams, or aggressive voices. |
| Mobile App Integration | Provide a mobile App where security personnel can monitor alerts in real-time. |
| IoT Integration | Connect the system with IoT security devices like alarms and motion sensors. |
| False Positive Reduction | Implement a confidence-based filtering system to minimize false alerts. |

SUMMARY

The “Suspicious Activity Simulator” is an advanced desktop application designed to detect acts of violence in real time using computer vision and machine learning. The system continuously analyzes live and recorded video feeds to identify violent behaviors and immediately notifies the relevant authorities. It utilizes YOLO v2 for recognizing violent actions and SSDMobileNetV2 COCO for identifying individuals involved in such incidents. By processing video frames and extracting key features, the system detects violence and triggers an alert system. These alerts are sent via Telegram API to the admin office for verification, ensuring that only genuine threats are escalated to law enforcement.

To maintain secure access and data integrity, the system employs authentication, allowing only authorized users to log in and manage surveillance activities. Additionally, a PHP-based backend facilitates seamless communication between the application and its database, ensuring efficient data handling. The system supports both live video feeds and recorded footage, providing a comprehensive security solution that allows for both real-time monitoring and retrospective analysis of past events. By integrating deep learning models and automated alert mechanisms, the system enhances security and ensures swift responses to potential threats.

The workflow of the system is structured to ensure a smooth and efficient response to violent incidents. First, user login, ensuring secure and authorized access. The system then processes video inputs from live camera feeds or recorded footage, analyzing each frame in real time. The YOLO v2 and SSDMobileNetV2 COCO models detect aggressive movements and identify individuals involved. Once a violent act is detected, an alert is triggered via Telegram API, instantly notifying the admin office.

The “Suspicious Activity Simulator” offers several advantages that make it an effective and scalable security solution. Real-time monitoring enables continuous surveillance, providing instant insights into ongoing situations in public spaces, workplaces, and institutions. Automated detection minimizes human intervention, reducing errors in manual surveillance and improving detection accuracy. The system’s ability to send instant alerts via Telegram API significantly improves response times, allowing authorities to take quick action before incidents escalate. Additionally, its scalable design allows deployment across multiple locations with minimal modifications, making it suitable for a wide range of environments, such as shopping malls, airports, universities, and corporate offices.

By integrating AI-driven violence detection, automated alert mechanisms, and real-time notifications, the “Suspicious Activity Simulator” provides a proactive approach to public safety. The combination of secure authentication, AI-powered surveillance, and automated alerts makes this system a robust and reliable security solution for modern safety challenges.

**Conclusion**

The ‘Suspicious Activity Simulator’ offers numerous advantages that make it an effective and scalable security solution. One of its key benefits is real-time monitoring, which provides continuous surveillance and enhances situational awareness. By automating violence detection, the system reduces human intervention, minimizing errors associated with manual surveillance. This increases detection accuracy and ensures that threats are identified as soon as they occur. Another critical advantage is its fast response time—by integrating the Telegram API, the system instantly notifies administrators, allowing them to verify alerts quickly and take necessary action. Additionally, the system’s scalability makes it suitable for various environments, including public spaces, educational institutions, corporate offices, and transportation hubs. It can be easily expanded to monitor multiple locations with minimal modifications, making it an adaptable solution for different security needs.

Beyond immediate threat detection, the system also plays a crucial role in crime prevention. By detecting violent incidents in their early stages, security personnel and law enforcement agencies can intervene before situations escalate. This proactive approach helps reduce crime rates and ensures public safety in high-risk areas. The system’s ability to store and analyze recorded footage also provides valuable insights for post-incident investigations, allowing authorities to gather evidence, identify suspects, and improve future security measures. The combination of AI-powered surveillance, automated alerts, and secure authentication makes this system a highly reliable and effective solution for modern security challenges.

In conclusion, the “Suspicious Activity Simulator”represents a significant advancement in AI-driven security solutions, offering an intelligent, automated, and scalable approach to violence detection. By integrating deep learning models, real-time video processing, and instant alert mechanisms, the system provides a robust framework for enhancing public safety. Its ability to detect violence in real time, minimize human error, and enable rapid responses makes it an essential tool for law enforcement, security personnel, and organizations seeking to improve their security infrastructure. As security threats continue to rise, implementing AI-powered surveillance systems like this one will play a crucial role in protecting lives, reducing crime, and ensuring a safer environment for everyone.

**REFERENCES**

* <https://www.youtube.com/watch?v=dQw4w9WgXcQ>
* <https://ieeexplore.ieee.org/abstract/document/9552469>
* https://www.mdpi.com/1999-5903/16/2/50
* <https://www.geeksforgeeks.org/>
* <https://www.youtube.com/watch?v=y1iXcCM5SLY>
* <https://github.com/topics/computer-science>
* <https://stackoverflow.com/questions/tagged/computer-science>
* https://www.youtube.com/watch?v=dQw4w9WgXcQ
* https://www.geeksforgeeks.org/
* https://www.youtube.com/watch?v=y1iXcCM5SLY
* <https://github.com/topics/computer-science>