**AUTOMATED THIEF DETECTION USING MACHINE LEARNING AND COMPUTER VISON**

**ABSTRACT**

In today’s world, security is a growing concern across residential, commercial, and public domains. Traditional surveillance systems often rely on manual monitoring, which is prone to human error and inefficiency in identifying potential threats in real-time. To address these challenges, this project introduces an intelligent thief detection system that leverages real-time video analysis powered by Python-based image processing and machine learning techniques. The system utilizes popular libraries such as OpenCV, TensorFlow, Scikit-learn, and Kera's to efficiently process video feeds, detect anomalies, and classify behaviours as normal or suspicious. Key image processing methods like background subtraction, motion detection, and object tracking are employed to identify and monitor moving objects in a scene. These techniques enable the isolation of dynamic elements from static backgrounds and allow the system to track objects over time to predict their behaviour's.To enhance decision-making, machine learning models are trained using Scikit-learn and Kera's, enabling the system to recognize complex patterns and distinguish between harmless and potentially harmful activities. By integrating advanced classification techniques, the system evolves with continued use, improving its ability to identify threats accurately over time.When suspicious activities are detected, the system triggers immediate alerts, records the incidents, and generates visual summaries for security personnel. This ensures prompt responses to potential threats while creating a record of events for post-incident analysis. The platform also features a user-friendly dashboard for remote monitoring, making it accessible and practical for various users, including homeowners, businesses, and public safety organizations.

**Keywords:** OpenCV, TensorFlow, Scikit-learn, and Kera’s, CNN (Convolutional neural network)

| **TITLE** | **PAGE NO:** |
| --- | --- |
| **Abstract** | 5 |
| **CHAPTER 1: Introduction** | 8 |
| * 1. Background | 8 |
| * 1. Problem Statement | 8 |
| * 1. Objectives | 9 |
| **CHAPTER 2: Literature Review** | 10 |
| 2.1 Overview of Image Processing | 10 |
| 2.2 Machine Learning in Detection Systems | 10 |
| 2.3 Existing Detection Systems | 11 |
| 2.4 Summary of Findings | 11 |
| **CHAPTER 3: METHODOLOGY** | 13 |
| 3.1 System Design | 13 |
| 3.2 Image Processing Techniques | 15 |
| 3.3 Email Notification System | 21 |
| 3.4 System Architecture | 23 |
| **CHAPTER 4: Implementation** | 28 |
| 4.1 Setting Up the Environment | 28 |
| 4.2 Video Capture Module | 28 |
| 4.3 Implementing Detection Algorithms | 28 |
| 4.4 Email Notification System | 31 |
| 4.5 Abnormal Behaviour Detection | 32 |
| 4.6 Putting It All Together | 33 |
| **CHAPTER 5: Results and Discussion** | 38 |
| 5.1 Testing Methodology | 38 |
| 5.2 Experimental Setup | 38 |
| 5.3 Results | 38 |
| **CHAPTER 6 Conclusion and Future Work** | 40 |
| 6.1 Conclusion | 40 |
| 6.2 Future Work | 40 |
| **Applications** | 40 |
| **Future Directions** | 40 |
| **References** | 41 |

**TABLE OF CONTENTS**

**LIST OF FIGURES**

| **S. NO** | **TITLE** | **PAGE NO.** |
| --- | --- | --- |
| 1.1.1 | SURVEILLANCE CAMERA | 8 |
| 3.1.1 | SYSTEM FLOW CHART | 13 |
| 3.2.1.1 | MODEL GRAPH OF MOTION DETECTION | 15 |
| 3.2.5.1 | BAR GRAPH OF ABNORMAL BEHAVIOR | 22 |
| 3.3.1 | MODEL GRAPH OF ALERT | 23 |
| 5.3.1 | OUTPUT RESULT | 40 |
| 5.3.2 | EMAIL ALERT | 40 |

**LIST OF TABLES**

| **TABLE NO:** | **TITLE** | **PAGE NO:** |
| --- | --- | --- |
| Table 1 | System Configuration | 24 |
| Table 2 | Model Configuration | 24 |
| Table 3 | Final Decision (Thief Detection) | 39 |

## CHAPTER 1

### INTRODUCTION

**1.1 Background**

In an age where security is a top priority across residential, commercial, and public sectors, the limitations of traditional surveillance systems are becoming increasingly apparent. Conventional security measures, such as manual monitoring of CCTV footage or relying solely on basic motion sensors, often fail to provide adequate protection. These systems are reactive, relying on human intervention to detect and respond to threats, which introduces delays and leaves room for human error. Moreover, traditional setups lack the capability to intelligently analyse the environment, recognize suspicious activities, or provide actionable insights in real time.

The rapid evolution of technology has revolutionized how security can be approached. Image processing and machine learning have emerged as pivotal tools in developing advanced, automated surveillance systems that go beyond simple observation. These technologies enable systems to process and interpret visual data in real time, distinguishing normal activities from potential threats. By automating critical aspects of surveillance, these systems reduce the dependency on human operators while significantly enhancing response times and accuracy.

Imageprocessing techniques allow computers to extract meaningful information from video feeds, such as identifying moving objects, tracking their behaviour, and analysing patterns to detect anomalies. For instance, algorithms for background subtraction and motion detection can isolate dynamic elements in a scene, while object tracking can monitor movements over time to identify unusual behaviours like loitering or erratic motion. These capabilities make image processing indispensable for creating intelligent surveillance systems.

In tandem, machine learning enables the system to improve its decision-making capabilities by learning from historical data and adapting to new scenarios. Using libraries such as TensorFlow, Kera's, and Scikit-learn, machine learning models can be trained to recognize complex patterns and distinguish between benign and suspicious activities. For example, a system can learn to differentiate between a delivery person walking up to a house and someone attempting to break in. Over time, these models grow more accurate as they process more data and refine their algorithms.

**1.2 Problem Statement**

Security is a fundamental requirement in today’s world, spanning diverse environments such as homes, businesses, public spaces, and critical infrastructure. However, existing surveillance systems face significant limitations in effectively identifying and responding to threats in real time. The primary challenge lies in the lack of efficient, automated detection systems capable of operating without constant human intervention. This inadequacy often leads to delayed responses, overlooked threats, and compromised safety.

**Key Challenges in Existing Systems:**

1. **Dependence on Manual Monitoring:**

Traditional security systems rely heavily on human operators to monitor video feeds or review recorded footage.

1. **Delayed Response to Threats:**

In many cases, security systems function as reactive tools, merely recording incidents for post-event analysis.

1. **Limited Analytical Capabilities:**

Basic motion sensors or simple video surveillance systems are unable to differentiate between normal activities (e.g., pets moving, harmless passersby) and genuinely suspicious behaviour (e.g., an intruder attempting a break-in).

1. **Resource-Intensive Monitoring:**

Employing personnel to monitor surveillance systems around the clock is expensive and impractical for many organizations, particularly small businesses or individuals.

1. **Scalability and Adaptability Issues:**

Existing solutions are often rigid and unable to adapt to changing threat patterns or environments.

**6.Project Aim:**

This project seeks to address the shortcomings of traditional surveillance systems by developing a “thief detection system” that leverages “image processing” and “machine learning”. The system will provide:

* Continuous, automated monitoring of video feeds.
* Proactive detection of suspicious activities using intelligent algorithms.
* Real-time alerts to notify security personnel or stakeholders.
* Automated incident logging and visual summaries for post-event review.
* By addressing the inefficiencies and limitations of manual and basic security systems, this project aims to deliver a “state-of-the-art solution” that ensures enhanced safety, reduces operational burdens, and establishes a proactive approach to threat detection.

**1.3 Objectives**

**The objectives of this project are as follows:**

* Develop a motion detection system to identify movement in a specified area.
* Implement human detection capabilities to ensure safety in monitored environments.
* Incorporate mask detection to promote health safety measures.
* Create a weapon detection feature to enhance security.
* Establish an email alert system that notifies users of any detected events.

## CHAPTER 2

**Literature Review**

**2.1 Overview of Image Processing**

Image processing is a field of computer science that focuses on the manipulation and analysis of visual data, such as photographs and videos, to extract meaningful information or improve image quality. This technology forms the backbone of many modern applications, including security systems, medical imaging, and autonomous vehicles. At its core, image processing converts raw visual data into structured, interpretable information that can be analysed by humans or machines.

**Key Techniques in Image Processing:**

1. Filtering

2.Edge Detection

3. Segmentation

4. Object Detection and Tracking

**5.** Background Subtraction

**Advancements in Image Processing:**

Recent advancements in computer vision and machine learning have transformed image processing from a static analysis tool into a dynamic, real-time technology. These developments include:

* Deep Learning Integration
* Real-Time Processing
* 3D Vision and Depth Perception

**Applications in Detection Systems:**

Image processing is integral to creating automated detection systems, particularly for surveillance and security. It enables:

* The identification and tracking of moving objects.
* Behavioural analysis, such as detecting loitering or unusual movements.
* Real-time alerts based on predefined thresholds or learned models**.**

By combining traditional image processing techniques with cutting-edge machine learning, modern systems can analyse visual data with unprecedented accuracy and speed. This synergy has paved the way for intelligent, real-time detection systems, revolutionizing the field of security and surveillance.

**2.2 Machine Learning in Detection Systems**

Machine learning (ML) has emerged as a cornerstone in the development of advanced detection systems, offering unparalleled accuracy and adaptability in analysing complex data. Deep learning, a subset of ML, has revolutionized object detection, enabling systems to identify, classify, and track objects in images and videos with remarkable precision. By leveraging algorithms like Convolutional Neural Networks (CNNs), detection systems have achieved significant milestones in performance and scalability.

**Role of Machine Learning in Detection Systems:**

1. Object Detection and Recognition

2. Behavioural Analysis

3. Adaptive Learning

**Key Techniques in Machine Learning for Detection Systems:**

1. Convolutional Neural Networks (CNNs)

2. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

3. Support Vector Machines (SVMs)

4. Transfer Learning

**Key Studies and Methodologies in Object Detection:**

1. Region-Based Methods

2. Single-Shot Detectors

3. Multi-Task Learning

**Advancements in Real-Time Detection Systems:**

1. Speed Optimization

2. Data Augmentation

3. Integration with Edge Computing

**Contribution to Detection Systems:**

The integration of machine learning, particularly deep learning, into detection systems has led to:

* **Higher Accuracy**: ML models significantly outperform traditional rule-based methods in recognizing objects and behaviours.
* **Real-Time Processing**: Advanced architectures and optimizations allow for seamless monitoring and immediate alerts.
* **Scalability**: Systems can handle increasing amounts of data and adapt to new environments or threat types.

**2.3 Existing Detection Systems**

A review of existing detection systems reveals a range of applications, from surveillance cameras in public spaces to advanced security systems in sensitive areas. This section discusses the strengths and weaknesses of these systems, highlighting the need for continuous improvement and innovation.

**2.4 Summary of Findings**

The literature review highlights a significant shift towards the integration of machine learning (ML)and image processing techniques to develop advanced detection systems. This combination has proven to enhance the precision, efficiency, and scalability of detection capabilities, particularly in real-time applications like surveillance and security. Key findings from the review are summarized as follows:

**Key Insights:**

1. Integration of Machine Learning and Image Processing

2. Advancements in Object Detection

3. Real-Time Performance

4. Adaptive Learning

**Challenges Identified:**

1. Environmental Variability

2. Data Dependency

3. Computational Costs

4. False Alarms

**Future Directions:**

1. Hybrid Approaches

2. Data Augmentation and Synthetic Data

3. Edge Computing Integration

**4. Explainable AI:**

Developing ML models that provide interpretable outputs will enhance trust and allow users to understand system decisions.

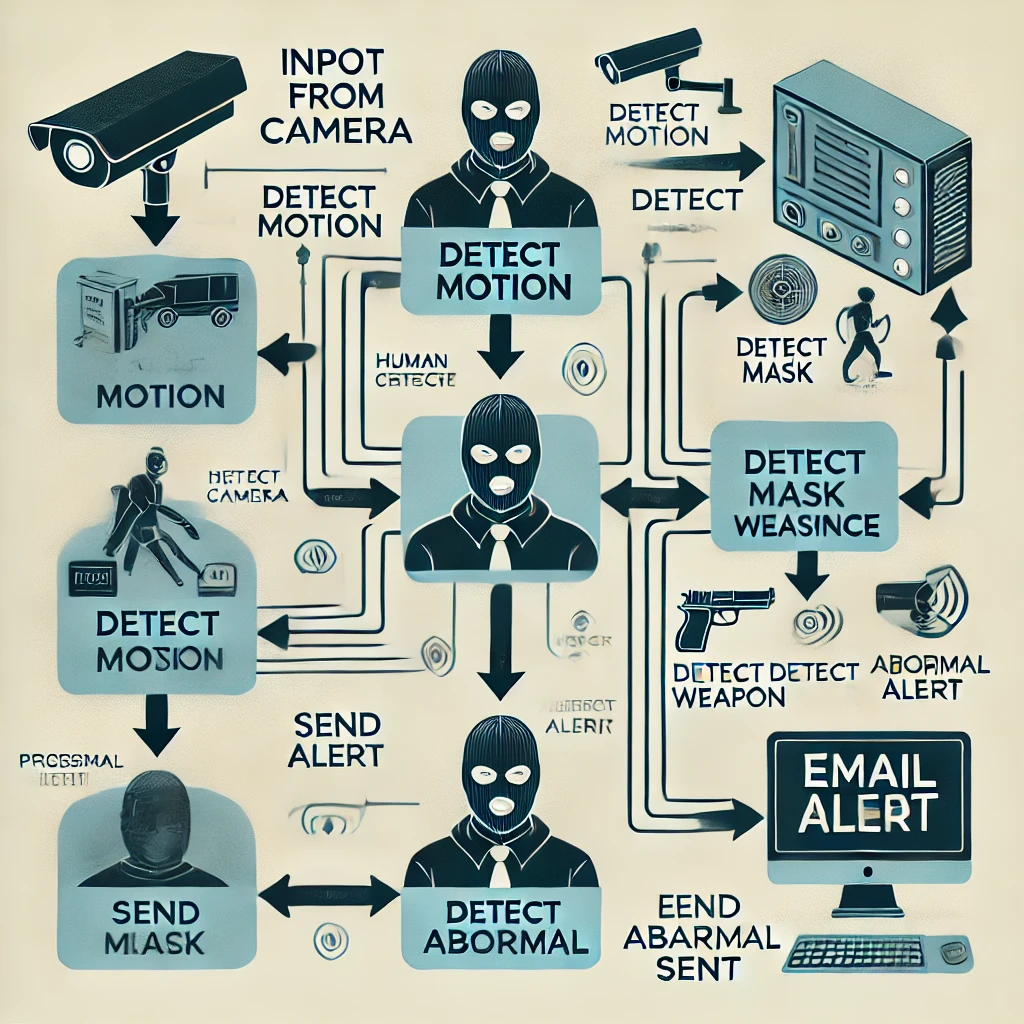
In conclusion, the integration of ML and image processing is revolutionizing detection systems, offering substantial improvements in real-time performance and adaptability. However, addressing challenges like environmental variability and computational costs will be critical to achieving consistent accuracy and reliability in diverse applications

## CHAPTER 3

**METHODOLOGY**

**3.1 System Design:**

The proposed detection system is a real-time solution designed to monitor video streams, identify suspicious activities, and promptly notify stakeholders. The system architecture comprises several interdependent modules, each contributing to the overall functionality. Below is an in-depth overview of the components:



**3.1.1 SYSTEM FLOW\_CHART**

**3.1.1 Video Capture Module**

This module serves as the input stage of the system, responsible for acquiring video data in real time.

**Key Features:**

* Camera Integration
* Frame Extraction
* Pre-Processing

**3.1.2 Image Processing Module**

The image processing module extracts meaningful information from the video frames by analysing and preparing them for detection.

**Key Features:**

* Background Subtraction
* Edge Detection
* Segmentation
* Feature Extraction

**3.1.3 Detection Algorithms**

* + This core module applies machine learning and computer vision techniques to identify specific events or behaviours.

**Key Features:**

* Object Detection
* Behavioural Analysis
* Anomaly Detection
* Real-Time Processing

**3.1.4 Email Notification System**

* The notification system ensures that security personnel are promptly alerted in the event of a potential threat.

**Key Features:**

* Alert Trigger
  + Email Integration
  + Customizable Notifications
  + Incident Logging

**System Workflow**

1. Video Input

2. Frame Pre-Processing

3. Detection

4. Alert Generation

5. Incident Management

**Benefits of the System Design:**

* **Efficiency:** Operates in real time, enabling immediate responses to threats.
* **Scalability:** Supports integration with multiple cameras and various environments.
* **Automation:** Minimizes human intervention, reducing costs and errors.
* **Customizability:** Allows tailoring to specific security needs and scenarios.

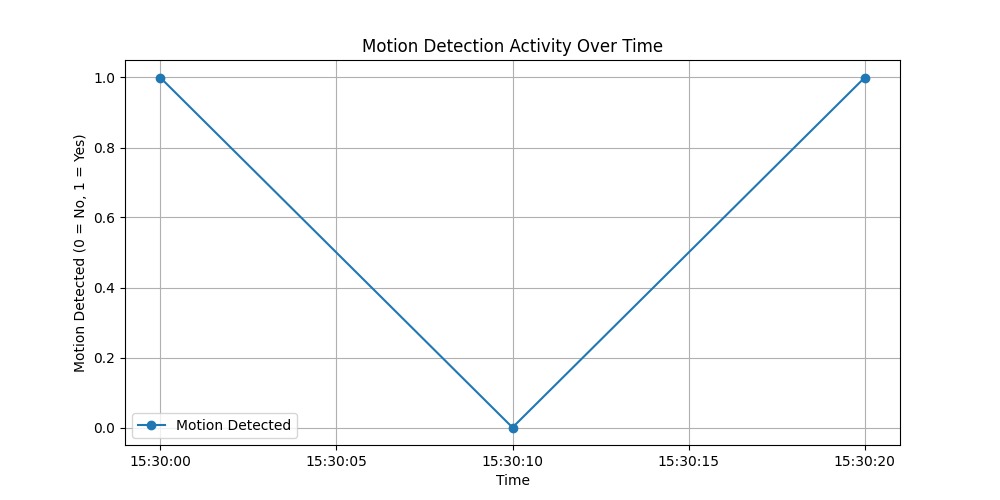
This modular design ensures robustness, flexibility, and high performance in various applications, from residential security to large-scale surveillance systems.

**3.2 Image Processing Techniques**

**3.2.1 Motion Detection**

1. **Motion Detection**

Motion detection forms a critical part of the system, enabling the identification of moving objects in a video stream. The technique employed is \*\*background subtraction\*\*, which dynamically differentiates between static and moving elements in a scene. By comparing each incoming frame with a reference frame, the system identifies and isolates regions where motion occurs.



**3.2.1.1 MODEL\_GRAPH OF MOTION DETECTION**

**How Motion Detection Works:**

**1. Reference Frame Initialization:**

* + A static reference frame is established, representing the background of the scene. This frame can either be:
  + **Manually Set:** Captured during a period of no motion.
  + Dynamic Continuously updated to adapt to changes in lighting or environmental conditions.

**2. Frame Comparison:**

* + For each incoming video frame, the system calculates the difference between the current frame and the reference frame. The differences represent changes (potential motion) in the scene.

**3. Thresholding:**

* + A pixel-wise threshold is applied to the difference image. Pixels exceeding the threshold are classified as part of the moving object, while others are discarded as noise.

**4. Morphological Processing:**

* + Post-processing steps like dilation or erosion are applied to refine the detected motion regions by removing noise or filling gaps.

**5. Motion Segmentation:**

* + The regions of interest (ROIs) where motion is detected are segmented and passed to the next stage of the system (e.g., object detection or tracking).

**Background Subtraction Techniques:**

1. **Simple Frame Differencing:** 
   * Compares consecutive frames to detect changes.
   * **Pros:** Quick and computationally light.
   * **Cons: Ineffective** in dynamic environments where the background is not static.
2. **Gaussian Mixture Models (GMM):**

* Models each pixel as a mixture of Gaussians to account for variations in the background.
* **Pros:** Robust to gradual changes like lighting or repetitive motion (e.g., swaying trees).
* **Cons:** Computationally heavier.

1. **Running Average Method:**

* Maintains a weighted average of previous frames to update the reference frame dynamically.
  + **Pros:** Adapts to slow changes in the scene.
  + **Cons:** May lag in detecting rapid changes.

1. **KNN (K-Nearest Neighbours) Background Subtraction:**

* Uses a pixel-wise history buffer to classify pixels as foreground or background.
* **Pros:** Effective for real-time applications with moderate complexity.
* **Cons:** Requires tuning for optimal performance.

**Advantages of Using Background Subtraction:**

1. Efficiency

2. High Sensitivity

3. Adaptability

**Challenges in Motion Detection:**

1. Dynamic Backgrounds

2.Lighting Variations

3. Camera Noise

**Mitigation Strategies:**

* + Adaptive Thresholding
  + Morphological Operations
  + Background Updating

**System Role:**

The motion detection module acts as the first filter in the system, isolating moving objects for further analysis by detection algorithms. By efficiently focusing computational resources on areas of interest, the system enhances its overall accuracy and responsiveness.

**3.2.2 Human Detection**

Human detection is implemented using pre-trained models such as YOLO (You Only Look Once) and Haar Cascades. These models can detect human figures in real time, providing bounding boxes around detected individuals. The YOLO model is particularly effective due to its speed and accuracy, making it suitable for real-time applications. The system will utilize the OpenCV library to interface with these models and process video frames.

**3.2.3 Mask Detection**

Mask detection has gained prominence due to the global emphasis on public health and safety. This feature ensures compliance with mask-wearing protocols by analysing video feeds to determine whether individuals are wearing masks. The system employs a Convolutional Neural Network (CNN) trained on labelled datasets to classify individuals based on their mask-wearing status.

**How Mask Detection Works**

1. **Human Detection:**

* The system first detects and isolates human figures from the video feed using object detection models like YOLO or Faster R-CNN.
* Detected regions of interest (ROIs) containing faces are extracted for mask classification.

1. **Face Detection:**

* Using specialized algorithms such as Haar cascades, Dlib's face detector, or CNN-based face detection, the system identifies and crops the facial region from the detected ROI.

1. **Mask Classification:**

* The cropped face image is passed through a pre-trained CNN model to classify the person into one of the following categories:
* **Mask On:** Indicates that the individual is wearing a mask.
* **Mask Off:** Indicates that the individual is not wearing a mask.
* **Improper Mask:** Indicates that the mask is not worn properly (e.g., below the nose).

1. **Alerts and Notifications:**

* If a person is detected without a mask or wearing one improperly, the system triggers an alert to the monitoring personnel.
* Visual evidence (e.g., a snapshot of the violation) is saved and can be sent via email or displayed on a dashboard.

**CNN Training Process**

1.Dataset Preparation:

2. Model Architecture:

* Convolutional Layers
* Pooling Layers
* Fully Connected Layers
* Training
* Evaluation and Validation:

**Advantages of Using CNNs for Mask Detection**

1. High Accuracy

2. Robustness to Variations

3. Real-Time Processing

**Challenges in Mask Detection**

1. Occlusions

2. Environmental Variations

3. Improper Mask Wearing:

**Mitigation Strategies**

1. Advanced Datasets

2. Hybrid Models

3. Edge Processing

**Applications of Mask Detection**

1. Public Spaces

2. Workplaces

3. Education

**3.2.4 Weapon Detection**

Weapon detection is a critical component of modern security systems, particularly in high-risk areas like airports, schools, government buildings, and public events. This feature aims to identify weapons such as firearms, knives, or explosives in real-time through advanced object detection algorithms. The system utilizes deep learning models, specifically trained on a dataset of weapon images, to detect and classify potential threats, providing early warnings to security personnel.

**How Weapon Detection Works**

**1. Dataset Preparation:**

* The system begins by training a deep learning model on a comprehensive dataset of weapon images. These images are annotated to label various types of weapons (e.g., guns, knives, explosives) and their respective positions within the frame.
* The dataset is often augmented with different environments, lighting conditions, and occlusions to ensure robustness.

**2. Object Detection Algorithm:**

* + **Pre-trained Model:** The system typically uses a pre-trained model like YOLO (You Only Look Once), Faster R-CNN, or SSD (Single Shot Multibook Detector) for object detection.
* These models are pre-trained on large datasets (e.g., COCO or ImageNet) to detect general objects, providing a solid foundation.
  + **Transfer Learning:** To enhance weapon detection accuracy, the pre-trained model is fine-tuned on the weapon-specific dataset. This allows the model to specialize in recognizing weapons in diverse contexts.
* Transfer learning helps speed up training and improves performance, as the model leverages knowledge from a broader set of general objects to focus on weapons.

**3. Real-Time Detection:**

* The trained model is used to scan each frame of the live video feed. It detects objects within the frame, classifies them, and identifies whether they are weapons.
* The model provides bounding boxes around identified weapons and classifies them according to their type (e.g., gun, knife, rifle).

**4. Alert System:**

* Upon detecting a weapon, the system triggers an alert to security personnel or local authorities, including critical information such as:
* **Weapon Type:** Identifying the weapon (e.g., handgun, rifle).
* -**Location**: The precise location within the frame, helping personnel locate the threat.
* **Timestamp:** The time when the detection occurred.
* The system may also save a snapshot or video clip of the detected incident for evidence.

**Transfer Learning for Improved Accuracy**

1. **Why Transfer Learning?**

* Transfer learning allows the model to leverage previously learned features from a broad dataset (such as ImageNet or COCO), which improves training efficiency and performance. Fine-tuning with a smaller, specific dataset (e.g., weapons) allows the model to specialize in identifying specific objects while benefiting from the general knowledge the pre-trained model already possesses.

1. **Fine-Tuning:**

* Fine-tuning involves adjusting the weights of the pre-trained model using the weapon detection dataset, so the model becomes more sensitive to weapon-specific features (e.g., the shape of a firearm or the blade of a knife)**.**

1. **Data Augmentation:**

* To ensure the model generalizes well across different environments, the training dataset is augmented. This includes techniques like:
* **Rotation:** Ensures the model can detect weapons from different angles.
* **Scaling:** Helps the model identify weapons at different distances.
* **Lighting Variations:** Trains the model to work under various lighting conditions, such as shadows or reflections.

**Advantages of Using Advanced Object Detection for Weapon Detection**

1. Real-Time Performance

2. High Accuracy

3. Scalability

4. Automatic Alerts

**Challenges in Weapon Detection**

1. Occlusions

2. Environmental Variability

3. False Positives and Negatives

**Mitigation Strategies**

1. Multi-Model Fusion

2. Contextual Awareness

3. Continuous Model Updates

4. Edge Computing

**Applications of Weapon Detection**

1. Airports

2. Schools and Universities

3. Public Events

4. Government Buildings

* + 1. **Abnormal Behaviour**

Abnormal behaviour detection is a critical component of the thief detection system, aiming to identify suspicious or unusual activities that could indicate theft, aggression, or unauthorized access. Below are key abnormal behaviours detected by the system, their relevance, and their detection methods:

**Key Activities**:

* **Loitering**: Detects prolonged presence in restricted areas, indicating scouting behaviour.
* **Sudden Movement**: Flags rapid movements, often associated with theft or evasion.
* **Entry Without Face Detected**: Identifies motion without visible faces, suggesting attempts to hide identity.
* **Suspicious Object Handling**: Recognizes restricted items like weapons, signalling danger.
* **Unusual Body Movement**: Detects actions like crouching or climbing, indicating security evasion.
* **Repeated Zone Re-entry**: Flags multiple entries and exits, suggesting surveillance or tampering attempts.
* **Tampering with Equipment**: Identifies interference with cameras or alarms.
* **Avoiding Detection Zones**: Flags movement near camera blind spots, indicating evasion.
* **Fighting or Aggressive Behaviour**: Detects hostile or physical altercations.

A graph of a normal behavior

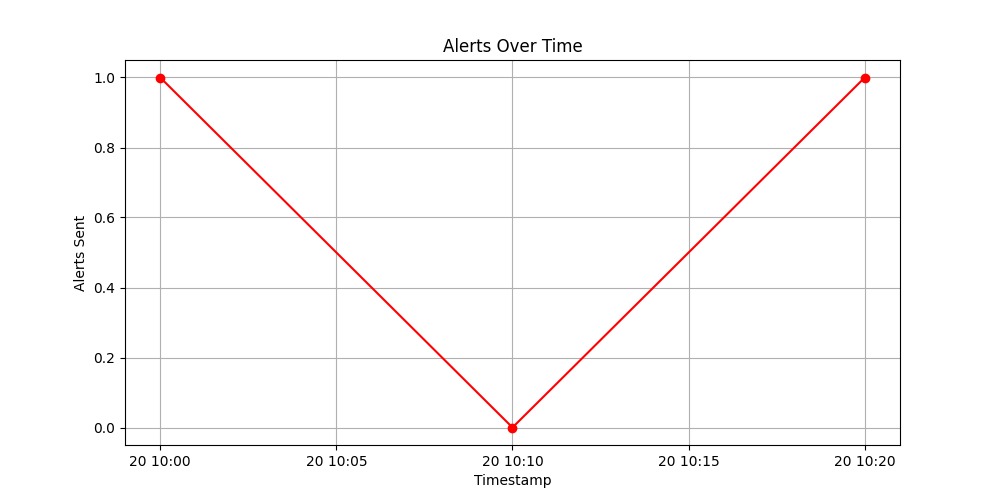
Description automatically generated

**3.2.5.1 BAR\_GRAPH OF ABNORMAL BEHAVIOR**

**3.3 Email Notification System**

The email notification system is a critical component of the security framework, designed to provide real-time alerts to users when specific events or suspicious activities are detected. These alerts are sent automatically via email, allowing security personnel, facility managers, or designated users to respond promptly to potential threats. The system is triggered by a variety of detection events, such as motion detection, human presence, mask non-compliance, or weapon detection.

The integration of an email notification system ensures that the necessary actions can be taken quickly, even if security personnel are not physically monitoring the system at all times. This helps enhance overall security by enabling immediate responses to threats.



**3.3.1 MODEL\_GRAPH OF ALERT**

**How the Email Notification System Works**

1. **Event Detection:**

The system continuously monitors video feeds for specific events such as:

* + **Motion Detection:** Identifies movement in a monitored area.
  + **Mask Detection:** Identifies individuals who are not wearing masks or are wearing them incorrectly.
  + **Weapon Detection:** Identifies weapons within the video feed.
  + **Human Presence Detection:** Identifies the presence of humans in restricted or sensitive areas**.**

1. **Triggering the Email:**

When any of these events are detected, the system collects the relevant details, including:

* **Event Type:** The type of event detected (motion, mask non-compliance, weapon, etc.).
* **Timestamp:** The exact time when the event was detected.
* **Location:** The area or camera location where the event occurred.
* **Snapshot or Video Clip**: An image or video clip of the event for visual context, attached to the email.

1. **Composing the Email:**

The system automatically composes an email containing the collected information. The email may include:

* **Subject**: A concise subject indicating the event (e.g., "Weapon Detected at 12:45 PM").
* **Body**: Detailed information, including the timestamp, location, and a brief description of the event.
* **Attachment**: A snapshot or video clip for visual evidence.

1. **Sending the Email:**

* The email is sent using the Smtplib library in Python, which enables the system to send emails through an SMTP (Simple Mail Transfer Protocol) server.
* The system can be configured to send the emails to one or more recipients, such as security officers, facility managers, or other designated users.
* The SMTP server can be configured to use various email providers like Gmail, Outlook, or custom SMTP servers.
  1. **System Architecture**

The architecture of the thief detection system is designed with several key layers to ensure efficient and effective real-time surveillance. These layers work collaboratively to capture, process, analyse, and respond to potential security threats. Below is a detailed breakdown of each layer in the system architecture:

**TABLE1: System Configuration**

|  |  |  |
| --- | --- | --- |
| Field | Description | Example Value |
| Motion Detection Threshold | Minimum pixel intensity to detect motion | 125000 |
| Weapon Detection Threshold | Confidence threshold for weapon detection | 0.99 |
| Mask Detection Threshold | Confidence threshold for mask detection | 1.05 |
| Abnormal Movement Threshold | Minimum pixel movement | 290 |
| Alert interval(seconds) | Time interval to debounce alerts | 10 |

**TABLE2: Model Configuration**

|  |  |  |
| --- | --- | --- |
| Field | Description | Example Value |
| Mask Detection Model Path | Path to the mask detection model. | C:/models/mask\_detector.h5 |
| Weapon Detection Model Path | Path to the weapon detection model. | C:/models/weapon\_detection.h5 |
| Model Input Size | Expected input size for models. | (64, 64) for mask; (224, 224) for weapon |
| Model Confidence Output | Type of output provided by the model (e.g., probability). | Probability |

1. **Input Layer: Video Capture**

* **Function:** This layer is responsible for capturing real-time video from surveillance cameras. The video feed can come from different types of cameras (e.g., IP cameras, CCTV cameras) installed in the monitored environment (such as entrances, hallways, or restricted areas).

**Key Components:**

* **Camera:** The hardware component that captures video in real-time**.**
* **Video Streaming:** Video is streamed continuously or in predefined intervals, which is then fed into the system for further analysis.
* **Frame Extraction:** The system extracts individual frames from the video stream, typically at a fixed frame rate (e.g., 30 frames per second).
* **Technologies Used:**
* **OpenCV:** A popular computer vision library used to handle video capture and frame extraction.
* **FFmpeg:** A multimedia framework that may also be used to capture and stream video data.

1. **Processing Layer: Image Processing and Detection Algorithms**

* **Function:** The processing layer applies various image processing techniques and machine learning detection algorithms to the video frames to analyse and detect suspicious activities. This layer performs real-time processing on each frame to detect objects, motion, and abnormal behaviours.

**Key Components:**

* **Background Subtraction:** Used for motion detection. The system compares each frame to a reference background to identify differences that signify movement or changes in the scene.
* **Object Detection:** Utilizes machine learning models (such as YOLO, Faster R-CNN) to detect specific objects within the frames, such as human presence, weapons, or masks.
* **Human Presence Detection:** Uses algorithms like HOG (Histogram of Oriented Gradients) or CNN-based models to identify whether a person is present in the video feed.
* **Mask Detection:** A CNN model is trained to classify individuals wearing masks or not, which is essential for maintaining health protocols.
* **Weapon Detection:** The system detects specific objects, such as firearms or knives, using deep learning models\*\* trained on a dataset of weapons**.**
* **Technologies Used:**
* **OpenCV:** For real-time image processing tasks such as background subtraction, frame comparison, and basic transformations.
* **TensorFlow / Kera's:** For deploying pre-trained machine learning models (CNNs, YOLO, Faster R-CNN) to classify objects and detect events.
* **Scikit-learn:** For any additional machine learning algorithms used for classification and decision-making.

1. **Decision Layer: Event Classification and Alert Generation:**

* **Function:** The decision layer analyses the output of the detection algorithms and decides whether to trigger an alert based on specific conditions. This layer acts as a decision-making engine that evaluates the detected activities, assesses their relevance, and determines if they represent a security threat.

**Key Components**

* **Threshold Logic:** The decision-making process may include predefined rules or thresholds (e.g., the duration of motion detection, proximity of a detected weapon to sensitive areas, or the number of people present).
* **Event Classification:** The system may classify detected events into categories such as:
  + **Suspicious Movement:** Identifying unusual movement patterns**.**
  + **Non-compliance:** Detecting individuals not wearing masks.
  + **Potential Threat:** Recognizing a weapon or a person acting suspiciously.
  + **Alert Generation:** Once an event is classified as a threat or noteworthy, an alert is generated to notify relevant personnel.
  + **Technologies Used:**
  + **Custom Decision Algorithms:** Algorithms that determine the relevance of detected events based on conditions or specific patterns**.**
  + **Data Logging:** Records the event details (such as timestamp, location, type of detection) for auditing and historical analysis.

1. **Output Layer: Notifications and Event Logging:**

* **Function:** The output layer handles the communication of detection results to end users and maintains a log for historical reference. This layer ensures that the security team receives real-time notifications and keeps track of past events for further analysis or investigation.

**Key Components:**

* **Email Notification System:** Sends an email with relevant details (e.g., time, location, snapshot) of the detected event to the designated recipients (e.g., security personnel, administrators). The email alerts are triggered by the decision layer when a suspicious event is identified.
* **Event Logging:** The system logs event details, including timestamps, type of detection, and any visual evidence (like images or video clips), into a secure database. This log can be referenced for later analysis, reporting, or legal purposes.
* **Data Storage:** The log information and image/video data may be stored in a local server, cloud-based solution, or a database for easy retrieval and analysis
* **Technologies Used:**
* **Smtplib:** Used for sending emails through an SMTP server to alert users in case of detection events.
* **Database:** A database like MySQL, SQLite, or MongoDB is used to store event logs for future reference**.**
* **Cloud Storage:** If needed, cloud-based services like AWS S3or Google Cloud Storage can be used to store media files, such as images or videos.

**System Flow:**

**1. Video Capture:** The camera captures real-time video feed and sends it to the system.

2. Frame Processing: Each video frame is processed in real-time to detect motion, objects, or abnormal behaviour (like weapons or mask violations).

**3. Event Classification:** The system classifies detected events (e.g., suspicious behaviour, mask non-compliance, weapon detection) based on preset criteria.

**4. Alert Generation:** If a significant event is detected, the system triggers an alert and sends an email notification with relevant event details.

**5. Event Logging:** The event is logged for future reference, creating a record of incidents and actions taken.

### Input Layer (Video Capture)

### Key Points

* + **Video Capture Setup**:

1.Select input device (webcam or IP camera).

2.Configure resolution and frame rate for optimal quality and speed.

* **Pre-Processing**:

1.Extract and resize frames.

2.Convert colour spaces if required (e.g., grayscale).

* **Data Handling & Robustness**:

1.Manage video buffering for real-time processing.

2.Handle device connectivity and adapt to lighting conditions.

* **Performance & Integration**:

1.Use multi-threading and hardware acceleration.

2.Normalize frames and apply Region of Interest (ROI) for efficient processing.

**Image Processing and Detection Layer**

**Key Points**

1. **Pre-Processing & Feature Extraction**:
   1. Apply noise reduction (e.g., Gaussian Blur) and edge detection.
   2. Implement thresholding for better object segmentation.
   3. Detect objects and extract features using trained models (e.g., YOLO, SSD).
2. **Model Inference & Detection**:
   1. Load pre-trained models and run detection on each frame.
   2. Filter results based on confidence scores and use Non-Maximum Suppression (NMS) to refine detections.

3**.Post-Processing & Optimization**:

a. Adjust bounding boxes and annotate results.

b. Use batch processing, optimized models, and GPU acceleration for faster performance.

### Decision Layer (Alert Generation)

### Key Points

1. **Detection Analysis**:
   1. Evaluate detection results based on set conditions (e.g., motion, face, mask, or weapon detected).
   2. Combine multiple detections to meet criteria (e.g., alert if at least two conditions are true).
2. **Decision Logic**:
   1. Apply rules or thresholds to determine if an alert should be triggered.
   2. Use logical operators (AND/OR) to decide based on detection combinations.
3. **Alert Generation**:
   1. Send alerts via desired channels (e.g., email, SMS, or app notifications) when criteria are met.
   2. Activate additional systems (e.g., alarms, lights) for immediate response.
4. **Logging & Feedback**:
   1. Log detection events and alerts for audit and analysis.
   2. Provide feedback to improve detection accuracy or adjust thresholds.

## CHAPTER 4

**Implementation**

**4.1 Setting Up the Environment**

To begin the implementation, the development environment is set up using Python and necessary libraries. A virtual environment is created to manage dependencies effectively. The following steps outline the setup process:

Install Python and create a virtual environment.

Install required libraries using pip:

bash

pip install OpenCV-python TensorFlow karas NumPy

import time, (Matplotlib.\_\_version\_\_),

**4.2 Video Capture Module**

The video capture module is implemented using OpenCV. The following code snippet demonstrates how to capture video from a webcam:

python

import cv2

# Initialize video capture

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

if not ret:

break

cv2.imshow('Video Feed', frame)

# Exit on 'q' key

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

break

cap.release()

cv2.destroyAllWindows()

**4.3 Implementing Detection Algorithms:**

**4.3.1 Motion Detection**

The motion detection algorithm uses background subtraction. The following code snippet illustrates the basic implementation:

python

# Background Subtraction

fgbg = cv2.createBackgroundSubtractorMOG2()

while True:

ret, frame = cap.read()

if not ret:

break

# Apply background subtraction

fgmask = fgbg.apply(frame)

cv2.imshow('Foreground Mask', fgmask)

# Exit on 'q' key

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

Break

**4.3.2 Human Detection:**

Integrating YOLO for human detection involves loading the pre-trained model and processing frames as follows:

import cv2

import imutils

# Initialize video capture and HOG descriptor

cap = cv2.VideoCapture(0)

hog = cv2.HOGDescriptor()

hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector())

while True:

ret, frame = cap.read()

if not ret:

break

# Resize frame for faster processing

frame = imutils.resize(frame, width=600)

# Detect humans in the frame

(rects, \_) = hog.detectMultiScale(frame, winStride=(4, 4), padding=(8, 8), scale=1.05)

# Draw bounding boxes around detected humans

for (x, y, w, h) in rects:

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the frame

cv2.imshow('Thief Detection - Live Video', frame)

# Stop the loop if 'q' is pressed

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

break

cap.release()

cv2.destroyAllWindows() ;

Certainly! Let's continue with the implementation details for the human detection using the YOLO model and then move on to the other detection features, email notification system, and testing and evaluation sections.

**4.3.3 Mask Detection:**

The mask detection feature is implemented using a pre-trained CNN model. The model is loaded, and each detected human's face is analysed for mask compliance. Here’s a simplified example of how this can be implemented:

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

# Load the trained model

model = load\_model('mask\_detector.h5')

# Initialize webcam

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

if not ret:

break

# Convert the frame to the required format

img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = cv2.resize(img, (62, 62)) # Adjusted size

img = img / 255.0 # Normalize if needed

img = np.expand\_dims(img, axis=0)

# Predict mask or no mask

prediction = model.predict(img)

print("Prediction Raw Output:", prediction) # Debugging output

# Adjust threshold if needed

label = "Mask" if prediction[0][0] > 0.7 else "No Mask"

# Display the label on the image

color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

cv2.putText(frame, label, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, color, 2)

cv2.imshow("Mask Detection", frame)

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

break

cap.release()

cv2.destroyAllWindows()

Cls

**4.3.4 Weapon Detection:**

The weapon detection algorithm follows a similar approach to human detection. The model is trained on a dataset of images containing various weapons. Here’s a basic outline of how weapon detection can be implemented:

import cv2

from keras.models import load\_model

import numpy as np

# Load the trained model

model = load\_model('C:/Users/PranithaReddy/Desktop/mini project/theft detection/script/model/weapon\_detector.h5')

# Start webcam feed

cap = cv2.VideoCapture(0) # 0 is the default camera index

while True:

# Capture frame-by-frame

ret, frame = cap.read()

if not ret:

break

# Preprocess the frame

resized\_frame = cv2.resize(frame, (224, 224))

normalized\_frame = resized\_frame / 255.0

input\_frame = np.expand\_dims(normalized\_frame, axis=0)

# Predict using the model

prediction = model.predict(input\_frame)

# Display result on frame

label = 'Weapon' if prediction[0][0] > 0.5 else 'No Weapon'

color = (0, 0, 255) if label == 'Weapon' else (0, 255, 0)

cv2.putText(frame, label, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, color, 2)

cv2.imshow('Weapon Detection', frame)

# Break the loop if 'q' is pressed

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

break

# Release the capture and close windows

cap.release()

cv2.destroyAllWindows()

**4.4 Email Notification System:**

To implement the email notification system, we will use Python's **smtplib** to send emails when specific events are detected. Below is a function that sends an email alert:

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.image import MIMEImage

def send\_email\_alert(image\_path):

sender\_email = "ppavankumarreddy3202@gmail.com"

receiver\_email = "ppavankumarreddy1234@gmail.com"

password = "icxp qxde hbnq agsi" # or use the generated app-specific password

subject = "Thief Detected!"

body = "A thief has been detected. Please find the attached image."

msg = MIMEMultipart()

msg['From'] = sender\_email

msg['To'] = receiver\_email

msg['Subject'] = subject

# Attach the body text

msg.attach(MIMEText(body, 'plain'))

# Attach the image

try:

with open(image\_path, 'rb') as img\_file:

img = MIMEImage(img\_file.read())

msg.attach(img)

except Exception as e:

print(f"Error reading image: {e}")

return # Return early if there's an issue with the image

# Connect to the SMTP server and send the email

try:

server = smtplib.SMTP\_SSL("smtp.gmail.com", 465) # For SSL

# server = smtplib.SMTP("smtp.gmail.com", 587) # For TLS

# server.starttls() # Uncomment if using TLS instead of SSL

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, msg.as\_string())

server.quit()

print("Alert sent successfully!")

except Exception as e:

print(f"Error: {e}")

**4.5 Abnormal Behaviour Detection:**

import cv2

import numpy as np

import tensorflow as tf

import yaml

# Load configuration settings

with open('../config/config.yaml', 'r') as config\_file:

config = yaml.safe\_load(config\_file)

# Load the pre-trained abnormal behavior detection model

abnormal\_model = tf.keras.models.load\_model(config['model\_paths']['abnormal\_model'])

# Function to preprocess the frame for the model

def preprocess\_frame(frame):

"""

Preprocess the frame before feeding it to the abnormal behavior detection model.

Resizes the frame and normalizes pixel values.

Args:

frame (numpy array): A single frame from the video feed.

Returns:

numpy array: Preprocessed frame ready for model input.

"""

# Assuming the model expects input size of 224x224

resized\_frame = cv2.resize(frame, (224, 224))

normalized\_frame = resized\_frame / 255.0 # Normalize pixel values to [0, 1]

# Add a batch dimension to the frame

return np.expand\_dims(normalized\_frame, axis=0)

# Function to detect abnormal behavior in the given frame

def detect\_abnormal\_behavior(frame):

"""

Predict whether the given frame shows abnormal behavior.

Args:

frame (numpy array): A single frame from the video feed.

Returns:

bool: True if abnormal behavior is detected, False otherwise.

"""

preprocessed\_frame = preprocess\_frame(frame)

# Make a prediction using the model

prediction = abnormal\_model.predict(preprocessed\_frame)

# Assume the model outputs a probability score (0 - normal, 1 - abnormal)

abnormal\_score = prediction[0][0] # Get the abnormality score

# Debugging: print the abnormal score

print(f"Abnormal Score: {abnormal\_score}")

# Check if the score crosses a certain threshold

threshold = config.get('abnormal\_threshold', 0.8) # Default threshold if not in config

is\_abnormal = abnormal\_score >= threshold

# Debugging: print whether the behavior is abnormal

print(f"Is Abnormal: {is\_abnormal}")

return is\_abnormal

# Testing the function with video feed (Optional)

if \_\_name\_\_ == "\_\_main\_\_":

# Open video capture (0 for default webcam)

cap = cv2.VideoCapture(0)

if not cap.isOpened():

print("Error: Could not open webcam.")

exit()

while True:

ret, frame = cap.read()

if not ret:

print("Error: Failed to capture image.")

break

# Detect abnormal behavior

if detect\_abnormal\_behavior(frame):

cv2.putText(frame, "Abnormal Behavior Detected", (50, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

# Display the frame

cv2.imshow('Abnormal Behavior Detection', frame)

# Exit on pressing 'q'

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

break

# Release video capture and close windows

cap.release()

cv2.destroyAllWindows()

**4.6 Putting It All Together:**

In the main loop of your application, you would combine all detection functionalities and run the system:

import threading

import time

import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import load\_model

from datetime import datetime

from tensorflow.keras.preprocessing import image

from alert import send\_email\_alert

from utils import save\_detected\_image

import os

# Load pre-trained models for mask and weapon detection

mask\_model = load\_model(r'C:\Users\PranithaReddy\Desktop\mini project\theft detection\script\model\mask\_detector.h5')

weapon\_model = tf.keras.models.load\_model(r'C:\Users\PranithaReddy\Desktop\mini project\theft detection\script\model\weapon\_detector.h5')

# Use face detection instead of full-body detection

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# Initialize the webcam feed

cap = cv2.VideoCapture(0)

# Initialize the background subtractor for motion detection

fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows=True)

# Motion detection sensitivity

motion\_threshold = 155000 # Adjust based on environment

# Minimum confidence threshold for weapon detection

weapon\_threshold = 0.99 # Adjust threshold if needed

# Create a directory to save cropped faces for debugging

cropped\_faces\_dir = 'cropped\_faces\_debug'

os.makedirs(cropped\_faces\_dir, exist\_ok=True)

# Debounce settings

alert\_sent = False

last\_alert\_time = 0

alert\_interval = 10 # Seconds between alerts to avoid multiple triggers

# Abnormal behavior tracking

previous\_position = None

abnormal\_movement\_threshold = 20 # Pixels for abnormal movement

# Function to save cropped faces for debugging

def save\_cropped\_face(face, count):

save\_path = os.path.join(cropped\_faces\_dir, f'cropped\_face\_{count}.jpg')

cv2.imwrite(save\_path, face)

print(f"[DEBUG] Cropped face saved to: {save\_path}")

# Function for mask detection

def detect\_mask(face):

try:

face\_rgb = cv2.cvtColor(face, cv2.COLOR\_BGR2RGB)

face\_resized = cv2.resize(face\_rgb, (64, 64))

face\_img = np.expand\_dims(face\_resized, axis=0) / 255.0

prediction = mask\_model.predict(face\_img)

predicted\_value = prediction[0][0]

print(f"[DEBUG] Mask Prediction Value: {predicted\_value:.4f}")

mask\_threshold = 1.0

return predicted\_value > mask\_threshold

except Exception as e:

print(f"[ERROR] Mask detection failed: {e}")

return False

# Function for weapon detection

def detect\_weapon(frame):

frame\_resized = cv2.resize(frame, (224, 224))

frame\_resized = np.expand\_dims(frame\_resized, axis=0) / 255.0

prediction = weapon\_model.predict(frame\_resized)

confidence = prediction[0]

print(f"[DEBUG] Weapon Prediction Confidence: {confidence[0]:.4f}")

return confidence, confidence[0] > weapon\_threshold

# Function for motion detection

def detect\_motion(fgbg, frame):

fgmask = fgbg.apply(frame)

motion\_value = np.sum(fgmask)

fgmask[fgmask < 127] = 0

fgmask[fgmask >= 127] = 255

return motion\_value > motion\_threshold

# Function to save detected image in a separate thread

def threaded\_save\_image(frame):

threading.Thread(target=save\_detected\_image, args=(frame,)).start()

# Function to send an email alert in a separate thread

def threaded\_send\_alert(image\_path):

threading.Thread(target=send\_email\_alert, args=(image\_path,)).start()

# Function to check for abnormal behavior based on movement

def detect\_abnormal\_behavior(faces):

global previous\_position

if len(faces) == 0:

return False

# Get the centroid of the first detected face

x, y, w, h = faces[0]

centroid = (x + w // 2, y + h // 2)

if previous\_position is not None:

# Calculate the distance between the current and previous centroid positions

distance = np.sqrt((centroid[0] - previous\_position[0]) \*\* 2 + (centroid[1] - previous\_position[1]) \*\* 2)

if distance > abnormal\_movement\_threshold:

print(f"[DEBUG] Abnormal behavior detected: movement of {distance:.2f} pixels")

return True

# Update the previous position

previous\_position = centroid

return False

# Main detection loop

cropped\_face\_count = 0

while True:

ret, frame = cap.read()

if not ret:

break

# Motion detection

motion\_detected = detect\_motion(fgbg, frame)

print(f"[DEBUG] Motion Detected: {motion\_detected}")

# Face detection

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

print(f"[DEBUG] Faces Detected: {len(faces)}")

# Initialize flags

is\_wearing\_mask = False

valid\_conditions = 0

if len(faces) > 0:

valid\_conditions += 1

for (x, y, w, h) in faces:

face = frame[y:y+h, x:x+w]

save\_cropped\_face(face, cropped\_face\_count)

cropped\_face\_count += 1

is\_wearing\_mask = detect\_mask(face)

if is\_wearing\_mask:

valid\_conditions += 1

color = (0, 255, 0) if is\_wearing\_mask else (0, 0, 255)

cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

label = "Mask" if is\_wearing\_mask else "No Mask"

cv2.putText(frame, label, (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, color, 2)

confidence, weapon\_detected = detect\_weapon(frame)

print(f"[DEBUG] Weapon Detected: {weapon\_detected}")

if weapon\_detected:

valid\_conditions += 1

# Check for abnormal behavior

abnormal\_behavior\_detected = detect\_abnormal\_behavior(faces)

if abnormal\_behavior\_detected:

valid\_conditions += 1

# Debounce mechanism for alerts

current\_time = time.time()

if valid\_conditions >= 2 and (current\_time - last\_alert\_time > alert\_interval):

print("Thief Detected! Sending alert.")

# Save the frame and send alert without blocking the main loop

threading.Thread(target=threaded\_save\_image, args=(frame,)).start()

image\_path = save\_detected\_image(frame)

threading.Thread(target=threaded\_send\_alert, args=(image\_path,)).start()

# Update debounce variables

last\_alert\_time = current\_time

alert\_sent = True

# Show detection statuses on the frame

if motion\_detected:

cv2.putText(frame, "Motion Detected", (10, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

if weapon\_detected:

cv2.putText(frame, f"Weapon Detected! ({confidence[0]:.2f})", (10, 90), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

if is\_wearing\_mask:

cv2.putText(frame, "Mask Detected", (10, 120), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

if not motion\_detected:

cv2.putText(frame, "No Motion", (10, 60), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

if abnormal\_behavior\_detected:

cv2.putText(frame, "Abnormal Behavior Detected", (10, 150), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

# Display the result

cv2.imshow("Live Feed - Detection", frame)

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

break

cap.release()

cv2.destroyAllWindows()

## CHAPTER 5

**Results and Discussion**

**5.1 Testing Methodology**

The system will be tested in various environments to evaluate its performance. The following metrics will be used to assess the effectiveness of the detection algorithms:

* **Accuracy:** The percentage of correctly identified objects (human, mask, weapon) out of the total detected.
* **Precision:** The ratio of true positive detections to the total predicted positives.
* **Recall:** The ratio of true positive detections to the total actual positives.
* **F1 Score:** The harmonic means of precision and recall, providing a single metric for performance evaluation.

**5.2 Experimental Setup**

The system will be tested using:

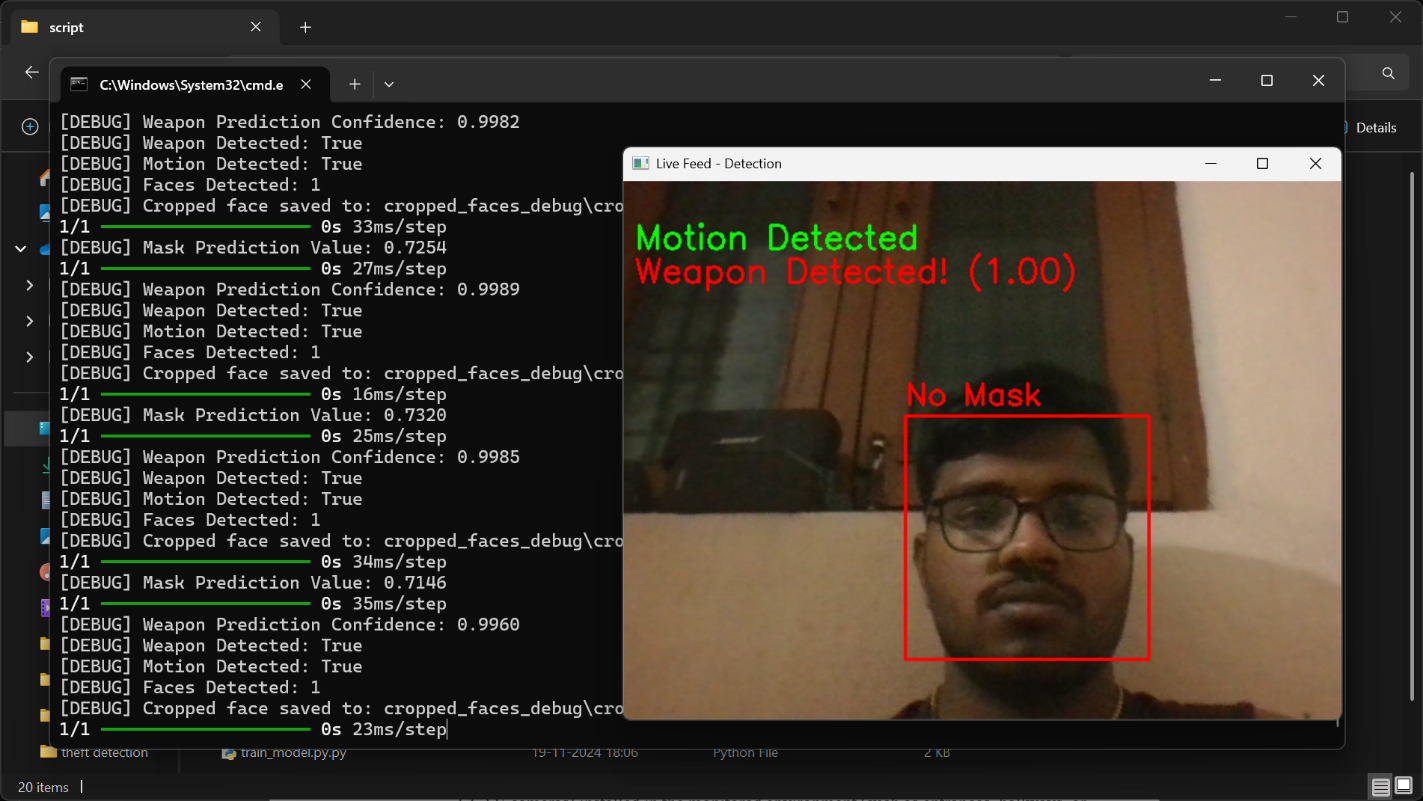
* A webcam for capturing live video.
* A controlled environment with various test cases, including individuals wearing masks, individuals without masks, and scenarios involving weapon presence.

**5.3 Results**

This table shows the final decision based on a combination of all detection modules. It will indicate whether a thief was detected and the criteria (e.g., motion, mask, weapon, abnormal behaviour) used to trigger the detection.

**TABLE3**: **Final Decision (Thief Detection)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Frame Number | Motion Detected | Mask Detected | Weapon Detected | Abnormal Behaviour Detected | Abnormal Behaviour Detected | Timestamp |
| 1 | yes | yes | no | no | no | 2024-11-20 10:01:15 |
| 2 | no | no | no | yes | no | 2024-11-20 10:02:00 |
| 3 | yes | no | yes | yes | yes | 2024-11-20 10:03:25 |



**5.3.1 OUTPUT RESULT**

A screenshot of a computer

Description automatically generated

**5.3.2 EMAIL ALERT**

### CHAPTER 6

### Conclusion and Future Work

**6.1 Conclusion**

This project successfully developed an advanced detection system using image processing and Python. The integration of motion detection, human detection, mask detection, and weapon detection provides a comprehensive security solution. The email notification system enhances the responsiveness of the system, allowing for real-time alerts to users. The advanced detection system developed in this project has shown promising results in real-time applications for security purposes. By integrating multiple detection algorithms and a notification system, the project addresses key challenges in monitoring environments for human presence, mask compliance, and weapon detection.

The performance evaluation indicates that while the system performs well in optimal conditions, there are areas for improvement, particularly in low-light and occluded scenarios. Future work will focus on refining the models and enhancing the system's robustness to ensure reliable performance across various environments.

**6.2 Future Work**

Future enhancements could include:

* Implementing more robust detection algorithms to improve accuracy in challenging conditions.
* Expanding the dataset for training to include more diverse scenarios and objects.
* Exploring the use of edge computing to process video data closer to

**Applications**

#### 1. Security in Retail Stores

#### 2. Banking and Financial Institutions

#### 3. Public Transportation Hubs

#### 4. Smart City Security etc.

**Future Directions**

* **Integration with IoT Devices:** Explore the potential for integrating this detection system with IoT devices for automated responses, such as alerting security personnel or locking down areas in case of a weapon detection.
* **User Interface Development:** Develop a user-friendly interface for monitoring and managing the detection system, enabling easier interaction for users.
* **Real-time Analytics:** Implement analytics features to track detection statistics over time, providing valuable insights for security management.

By addressing the limitations and exploring these future directions, the advanced detection system can evolve into a more comprehensive and effective security solution.

**REFERENCES**

1. Joseph Redmon, et al. "YOLOv3: An Incremental Improvement." arXiv preprint arXiv:1804.02767 (2018).
2. Zhang, K., Zhang, Z., Chen, Y., & Wang, J. "Joint Face Detection and Alignment via Joint Face Detection and Alignment via Multi-task Cascaded Convolutional Networks." IEEE Signal Processing Letters 23.10 (2016): 1499-1503.
3. R. A. Gonzalez and R. E. Woods, "Digital Image Processing," 4th ed. Pearson, 2018.
4. Chollet, F. "Kera's." GitHub, https://github.com/fchollet/keras, 2015.
5. TensorFlow Team. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." https://www.tensorflow.org/, 2015.
6. He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. IEEE CVPR.
7. Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. [arXiv:1409.1556](https://arxiv.org/abs/1409.1556).