AI-Powered Weapon Detection with Automated Alert System

This project report is submitted in the partial fulfillment of the requirements for the Award of the Degree

BACHELOR OF TECHOLOGY

IN

COMPUTERE SCIENCE AND ENGINEERING

Submitted by

SURAGANI VENKATA PRAVEEN

Reg.No: 218297601026

Under the esteemed guidance of

Mrs. P S V D Gayatri

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AKNU COLLEGE OF ENGINEERING

ADIKAVI NANNAYA UNIVERSITY, RAJAMAHENDRAVARAM

2021-2025

ADIKAVI NANNAYA UNIVERSITY

AKNU COLLEGE OF ENGINEERING

RAJAMAHENDRAVARAM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this internship report entitled, "**AI-Powered Weapon Detection with Automated Alert System**" is a Bonafide work of **SURAGANI VENKATA PRAVEEN**, Reg.No:218297601026 submitted in a partial fulfilment of the requirements for the award of Degree of BTech (CSE) during the period 2021-2025. This work carried out by him under my supervision and guidance and submitted to the Department of Computer Science and Engineering, Adikavi Nannaya University.

INTERNAL GUIDE **HEAD OF THE DEPARTMENT**

EXTERNAL EXAMINER

DECLARATION

I **SURAGANI VENKATA PRAVEEN,** Reg.no:218297601026 here by declare that

a. The work contained in this report is original and has been done by me under the guidance of **Mrs. P S V D Gayatri, Assistant Professor**.
b. The work has not been submitted to any other Institute for any degree or diploma.
c. I have followed the guidelines provided by the university in preparing the report.
d. I have confirmed the norms and guidelines given in the Ethical Code of Conduct of the Institute.
e. Whatever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

and submitted under the guidance of **Mrs. P S V D Gayatri, Assistant Professor**, Adikavi Nannaya University, for the partial fulfilment of requirements of the award of the degree, Bachelor of Technology in Computer Science and Engineering in the academic year 2021-2025.

Signature of the student

S.V. PRAVEEN

ACKNOWLEDGEMENT

I feel fortunate to pursue my **Bachelor of Technologies** degree in the **Department of Computer Science and Engineering, University College of Engineering, Adikavi Nannaya University**. It provided all the facilities in the area of Computer Science and Engineering. It's a genuine pleasure to express my deep sense of thanks and gratitude to my mentor and project guide **Mrs. P S V D Gayatri, Assistant Professor, Department of Computer Science and Engineering, Adikavi Nannaya University** for her excellent guidance right from the selection of the project and her valuable suggestions throughout the project work. Her constant and timely counsel has caused me to succeed in completing this project in college.

I profusely thank **Dr. P. Venkateswara Rao**, Principal, University College of Engineering, for all the encouragement and support.

I also thank **Dr. B. Kezia Rani**, Head of the Department of Computer Science and Engineering, for the guidance.

I also thank **Dr. D. Latha**, Project Coordinator, Department, Computer Science and Engineering, for the guidance.

A great deal of thanks goes to Review Committee Members and the entire Faculty of the Department of Computer Science and Engineering, University College of Engineering, Adikavi Nannaya University, Rajamahendravaram for their excellent supervision and valuable suggestions.

SURAGANI VENKATA PRAVEEN

218297601026

ABSTRACT

Real-time weapon detection has become essential for preventing threats in public and private spaces. This project uses YOLOv11, a powerful object detection model, to identify weapons like knives and guns in live surveillance(camera) footage. When a weapon is detected, the system sends a WhatsApp message to security personnel and it saves the image when the new weapon is detected, the alert message will specify weapon it was detected along with the image of weapon detected, allowing security persons to take immediate action, this will prevent the attack. This technology is useful in airports, schools, malls, banks, and public events to prevent crimes and enhance safety. By combining deep learning with real-time WhatsApp alerts, the system ensures a fast and effective response to potential threats.

Keywords— Smart Surveillance, YOLOv11, WhatsApp Alert Message, Object detection, Attack prevention.

INDEX

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Automated surveillance solutions offer the ability to enhance both the efficiency and accuracy of monitoring systems by detecting Weapons like knives and guns and providing real-time analysis without continuous human oversight. YOLOv11 (You Only Look Once), a state-of-the-art object detection algorithm, has emerged as a key technology in this domain due to its real-time processing capabilities and high accuracy in identifying objects and behaviors within crowded environments. This technology can significantly improve the detection of Weapons and prevent attacks. Moreover, the integration of AI with existing closed-circuit television (CCTV) infrastructure introduces new possibilities for proactive security management. Rather than merely recording events for later review, AI-powered systems analyze live video streams, detecting potential threats from Weapons and anomalies as they occur. This evolution from traditional surveillance to intelligent, real-time monitoring systems has the potential to redefine how public safety is managed in large, dynamic environments.

Using the Pywhatkit library integrates the WhatsApp alert message To the weapon detection whenever a new weapon is detected, the WhatsApp alert message will send to the concerned person with the image of the weapon detected, the type of weapon detected like guns and knives and the count of the weapons. The system captures the video frames from the CCTV cameras and then each frame is sent to YOLOv11 model to detect the weapon, Yolov11 architecture allows fast and efficient detection with computational overhead. When a weapon is detected the system automatically sends a WhatsApp alert message to the security person concerned. one of the key features of the system is instant alert message. This AI-driven surveillance system can transform traditional CCTV setups into intelligent security networks, enabling proactive threat detection rather than just passive monitoring. Unlike conventional systems that require manual intervention for analysis, this solution provides automated threat detection, real-time notifications, and improved situational awareness. The integration of deep learning with security surveillance ensures that potential threats are identified with high accuracy and responded to promptly, ultimately enhancing public safety. By utilizing YOLOv11 for real-time detection and WhatsApp alerts for instant communication, this project aims to bridge the gap between AI-powered security

monitoring and immediate response mechanisms. The proposed system is scalable and can be implemented across various sectors where security is a priority.

1.1Problem Statement:

Integrating Artificial Intelligence and Machine Learning (AI/ML) technologies into the extensive network of CCTV cameras positioned throughout train stations and tracks may be easily accessed by AI/ML systems. These systems use YOLOv11 to examine each real-time frame from the CCTV feeds and identify anyone acting suspiciously Carrying Weapon. Weapon Detection: To identify people carrying knives or Guns, YOLOv11 will detect the weapons in live. The device instantly notifies on the display if a weapon is detected to the security guard in the CCTV control room. At the same time, it Send the alert to the security or the police for quick action and response

1.2 Existing System:

Currently, most security systems rely on manual monitoring of CCTV cameras by security personnel, which is time-consuming, prone to human error, and inefficient in detecting threats quickly. Some automated security systems use traditional motion detection or metal detectors, but these methods have limitations:

1. **Manual CCTV Surveillance** – Security personnel must continuously monitor live footage, which can lead to missed threats due to distractions.
2. **Metal Detectors & X-ray Scanners** – These are only effective in controlled environments like airports but cannot detect concealed weapons in open areas.

Due to these limitations, existing systems are not efficient in real-time weapon detection and response, leading to delays in security actions. This project improves upon these drawbacks by using YOLOv11 for accurate detection and sending instant WhatsApp alerts, ensuring a faster and more effective response to threats.

1.3 Proposed System:

The aim is to overcome the limitations of existing security systems by using YOLOv11, a powerful deep-learning model, for real-time weapon detection and an instant alert mechanism via WhatsApp. The system works as follows:

1. Weapon Detection using YOLOv11 – The model is trained on a large dataset of weapons like knives and guns to improve accuracy. It can detect weapons in live video feeds from CCTV cameras.
2. Real-time Processing – The system processes video frames quickly, ensuring instant identification of weapons without delays.

3. WhatsApp Alert Notification – When a weapon is detected, an automated WhatsApp message is sent to security personnel, specifying which weapon was detected along with the image of weapon detected.
4. Enhanced Security Response – By providing real-time alerts, security teams can respond faster, reducing the risk of potential attacks.

Advantages of the Proposed System

- Faster Threat Detection – Detects weapons in real-time, reducing human effort.
- Accurate & Reliable – Uses advanced AI to improve detection accuracy
- Instant Alerts via WhatsApp – Ensures security personnel are notified immediately.
- Scalable for Various Locations – Can be implemented in airports, schools, malls, banks, and public events for enhanced safety.

By integrating AI-powered detection with instant messaging alerts, this system provides a smart and efficient security solution for preventing potential threats.

CHAPTER 2
LITERATURE SURVEY

In recent years, AI-powered smart surveillance systems have attracted a lot of interest due to their potential to improve crowd dynamics and public safety. In order to identify suspicious activity, stop crime, and improve crowd management in real time, these systems make use of cutting-edge technology like computer vision, machine learning, and predictive analytics. The main conclusions, difficulties, and uses of current studies on AI-driven surveillance in crowd control and crime prevention are summarized in this review of the literature. AI-powered surveillance has shown significant promise in the field of crime prevention in terms of proactive law enforcement actions and real-time Weapon detection.

2.1 Hybrid AI Surveillance: Advanced Techniques in Object Detection, Behavior Analysis, and Action Recognition [3]: The survey explores a number of aspects related to surveillance applications, such as object detection, facial identification, anomaly detection, behavior analysis, and scene interpretation. One may think of the suggested security camera system as a hybrid twin system. We experimented with and assessed an action recognition technique using time-series inference skeleton information, and we carried out skeleton extraction on an edge AI processor and assessed it experimentally in terms of processing performance and power consumption

2.2 AI-Driven Cybersecurity: Scalable Solutions for Rapid Intrusion Detection [5]**:** Artificial intelligence (AI) technologies have been used to collect data, which can subsequently be analyzed to produce knowledge that is useful for preventing intrusions. The scalability of these AI-based cybersecurity frameworks is impressive, and they can identify criminal activity in cyberspace faster and more effectively than traditional security architecture.

2.3 Enhancing Crime Prevention and Crowd Management [2]**:** This essay will look at how supervision, data analysis, and cyber security help prevent the deployment of AI and IoT technology. The study also discusses the advantages of combining AI with IoT technology, as well as the difficulties and moral dilemmas that arise. Furthermore, the

application of artificial intelligence (AI) and the Internet of Things (IoT) to the prevention of crimes against states is reviewed, along with suggestions for additional research

2.4 A Hybrid Twin System Driven by Edge AI for Skeletal Data-Based Real-Time Action Recognition [9]**:** One may think of the suggested security camera system as a hybrid twin system. We experimented with and assessed an action recognition technique using time-series inference skeleton information, and we carried out skeleton extraction on an edge AI processor and assessed it experimentally in terms of processing performance and power consumption.

2.5 Detecting Suspicious Behavior on Surveillance Videos [7]**:** In order to concentrate on suspicious behavior prior to a crime occurring, the Precrime Behavior technique eliminates information pertaining to criminal commission. The generated samples from various crime types are visually comparable to samples of normal conduct. We deployed 3D Convolutional Neural Networks and trained them using several methodologies to solve this issue.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1Hardware Requirements:

1. **Processor: Intel Core i5/i7**

   A high-performance processor is required for real-time video processing and deep learning model execution.

2. **Hard Disk: 256 GB SSD**

   A Solid-State Drive (SSD) is preferred for faster data processing and model loading.

3. **Monitor: 15’’inch**

   A clear and responsive display is necessary for monitoring detection results.

4. **RAM: 8 GB**

   A minimum of 8 GB RAM ensures smooth performance while running YOLOv8 and handling video streams.

5. **GPU: NVIDIA RTX 3060**

   A dedicated GPU is essential for running YOLOv8 efficiently, as deep learning models require parallel processing.

6. **CCTV Cameras**

   The system integrates with CCTV cameras to capture live footage for weapon detection.

3.2Software Requirements:

1. **Operating System: Windows 11**

   The system is compatible with Windows and Linux-based OS for better flexibility in deployment.

2. **Programming Languages: Python, HTML, CSS, JavaScript**

   - **Python**: Used for implementing the YOLOv8 model and integrating WhatsApp API.
   - **HTML, CSS, JavaScript**: Used for creating a web-based dashboard for monitoring detections.

3. **Frameworks & Libraries:**

   - **OpenCV** – For video processing.
   - **PyTorch/TensorFlow** – For YOLOv11 implementation.
   - **Ultralytics YOLOv11** – pre-trained YOLOv11 model for object detection.

- **Pywhatkit** – For sending WhatsApp alerts.
- **Deep_sort** – For tracking the weapons.
- **Flask** – to deploy the model to connect with website.

4. **Database: MySQL / Firebase**
    - Stores detection logs and historical records for security teams.
    - Stores detected weapons images.
5. **GoggleColab**:
    - We can train the model on the google colab easily with the access of GPUs.
    - We can train and test the model

3.3 Functional Requirements:

3.3.1 Data Input and Collection

- The system captures real-time video feeds from CCTV cameras.
- Frames are extracted and processed using YOLOv11 to detect weapons.
- The detected weapon type is logged into the database.

3.3.2 Data Processing and Analysis

- YOLOv11 model is used to classify objects as weapons (e.g., knife, gun, etc.).
- Bounding boxes and confidence scores are assigned to detected weapons.
- If a weapon is detected, an automated WhatsApp alert is triggered.

3.3.3 Alert Notification System

- Once a weapon is detected, the system sends:
    - A WhatsApp message to security personnel.
    - Details of the detected weapon (e.g., "Gun detected along with image").

3.4 Non-Functional Requirements:

3.4.1 Real-time Performance

- The system processes video frames in real-time with minimal latency.
- Optimized GPU processing ensures quick weapon detection.

3.4.2 Time-Efficient Data Processing

- YOLOv11 ensures fast object detection with a high FPS (Frames Per Second) rate.
- Efficient message handling via WhatsApp API ensures instant alerts.

3.4.3 Availability & Reliability

- The system is available 24/7, ensuring continuous monitoring.
- Failsafe mechanisms ensure that alerts are always sent in case of a threat.

3.5 Dataset Used:

The Guns-Knives Object Detection Dataset from Kaggle is a collection of images specifically designed for training and evaluating object detection models for recognizing weapons, particularly guns and knives. The dataset is structured to facilitate training deep learning models like YOLO (You Only Look Once) for real-time weapon detection in various environments.

Each image has a corresponding annotation file in YOLO format, where:

- Each line in the annotation file represents an object detected in the image.
- The format of the annotation is:

<class_id> <x_center> <y_center> <width> <height>

  - class_id: The label of the object (0 for knife, 1 for gun, etc.).
  - x_center, y_center: The normalized center coordinates of the bounding box.
  - width, height: The normalized width and height of the bounding box.

Classes in the Dataset

The dataset is labelled for two primary object categories:

1. **Knife –** labelled as 0
2. **Pistol –** labelled as 1

CHAPTER 4
SYSTEM DESIGN

4.1 System Architecture:

A Weapon Detection System is designed to enhance security in public places, leveraging AI-based real-time object detection to identify weapons and trigger instant alerts via WhatsApp. The system architecture consists of multiple components that work together to ensure accurate detection, efficient processing, and quick notifications. Below is an overview of the key aspects of the system architecture:

1. User Interface (UI)

- **Purpose:** Provides a user-friendly dashboard for security personnel to monitor weapon detections.
- **Components:** Web-based dashboard application.
- **Features:**
    - Live video feed with bounding boxes for detected weapons.
    - Weapon detection class and confidence scores.
    - Alert history and status tracking.

2. Data Ingestion

- **Purpose:** Captures and processes video footage from surveillance cameras.
- **Components:**
    - **CCTV cameras (IP/HD Cameras)** – Used for live streaming.
    - **Frame extraction module** – Captures frames from the video stream.
    - **Pre-processing module** – Enhances images for better detection accuracy.
- **Integration**: Ensures real-time video streaming and frame processing.

3. Data Processing and Analysis

- **Purpose:** Analyses video frames to detect weapons using YOLOv11.
- **Components:**
    - **YOLOv11 Model** – Pre-trained deep learning model for object detection.
    - **OpenCV** – For image processing and video frame extraction.
- **Tasks:**
    - Identifies and classifies weapons (e.g., knife, gun, pistol).
    - Assigns bounding boxes and confidence scores to detected objects.

- Sends detection results for further processing.

4. Decision Support System (Alert Mechanism)

- **Purpose:** Triggers alerts when a weapon is detected.
- **Components:**
  - **Pywhatkit (WhatsApp Alerts)** – Sends automated security alerts.
  - **Notification System** – Class detections and sends alerts to security personnel.
  - **Database (MySQL/Firebase)** – Stores detection weapon image.
- **Output:**
  - WhatsApp alert message with weapon details and image.
  - Security team notified in real-time for immediate action.
  - Detection logs stored for future reference.

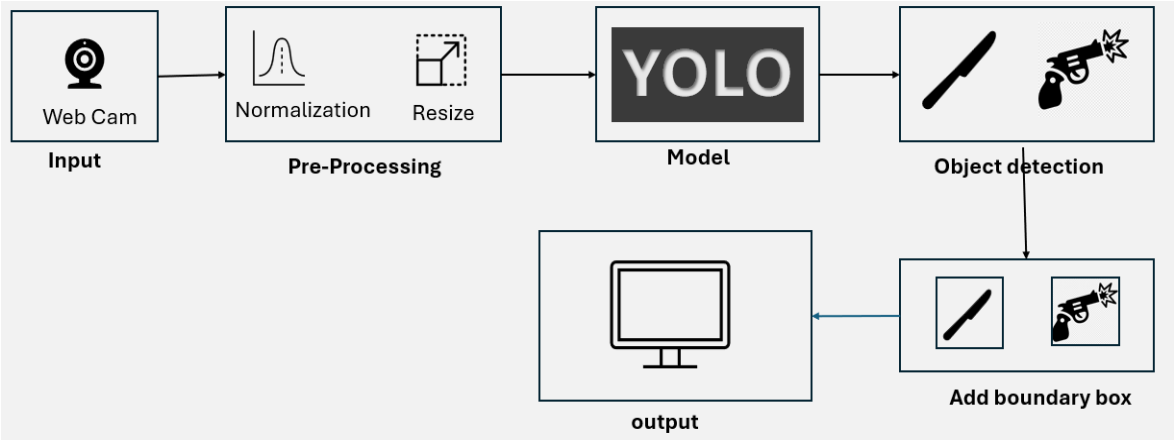

Fig 4.1: System Architecture

This architecture ensures a scalable and efficient weapon detection system by integrating AI-based real-time detection, instant messaging alerts, and user-friendly monitoring the security.

CHAPTER 5
UML DIAGRAMS

UML is simply another graphical representation of a common semantic model.

UML provides a comprehensive notation for the full lifecycle of object-oriented development.

ADVANTAGES:

• To represent complete systems using object-oriented concepts

• To establish an explicit coupling between concepts and executable code

• To take into account the scaling factors that are inherent to complex and

• Critical end.

UML defines several models for representing systems :

• The class model captures the static structure

• The state model expresses the dynamic behavior of objects

• The use case model describes the requirements of the user

• The interaction model represents the scenarios and messages flows

• The implementation model shows the work units

• The deployment model provides details that pertain to processing.

5.1 USECASE DIAGRAMS:

Use case diagrams and overview the usage requirement for the system. They are useful for presentations to management and/or project stakeholders, but for actual development you will find that use cases provide significantly more value because they describe “the meant” of the actual requirements. A use case describes a sequence of actions that provides something of measurable value to an action and is drawn as a horizontal ellipse.

Actors:

1. **System (Left Side)**: Represents the weapon detection software.
2. **User (Right Side)**: Represents the end-user (security personnel, admin, or law enforcement).

Actions:

1. **Start Web Cam:** The system initializes the webcam or CCTV When User Starts it.

2. **Preprocess Image:** The captured frames are processed for better detection accuracy.
3. **Training Model**: The system trains the YOLOv11 model on a dataset of weapons (guns, knives, pistols, etc.) to improve detection.
4. **Feature Extraction:** Important features (edges, contours, textures) are extracted to identify the weapon.
5. **Detect Class:** The system classifies the weapon when it detected.
6. **Add Boundary Boxes:** Once a weapon is detected, a bounding box is drawn around it with a confidence score.
7. **Display Output:** The final detection result is displayed to the user with real-time annotations, and an alert message is triggered.

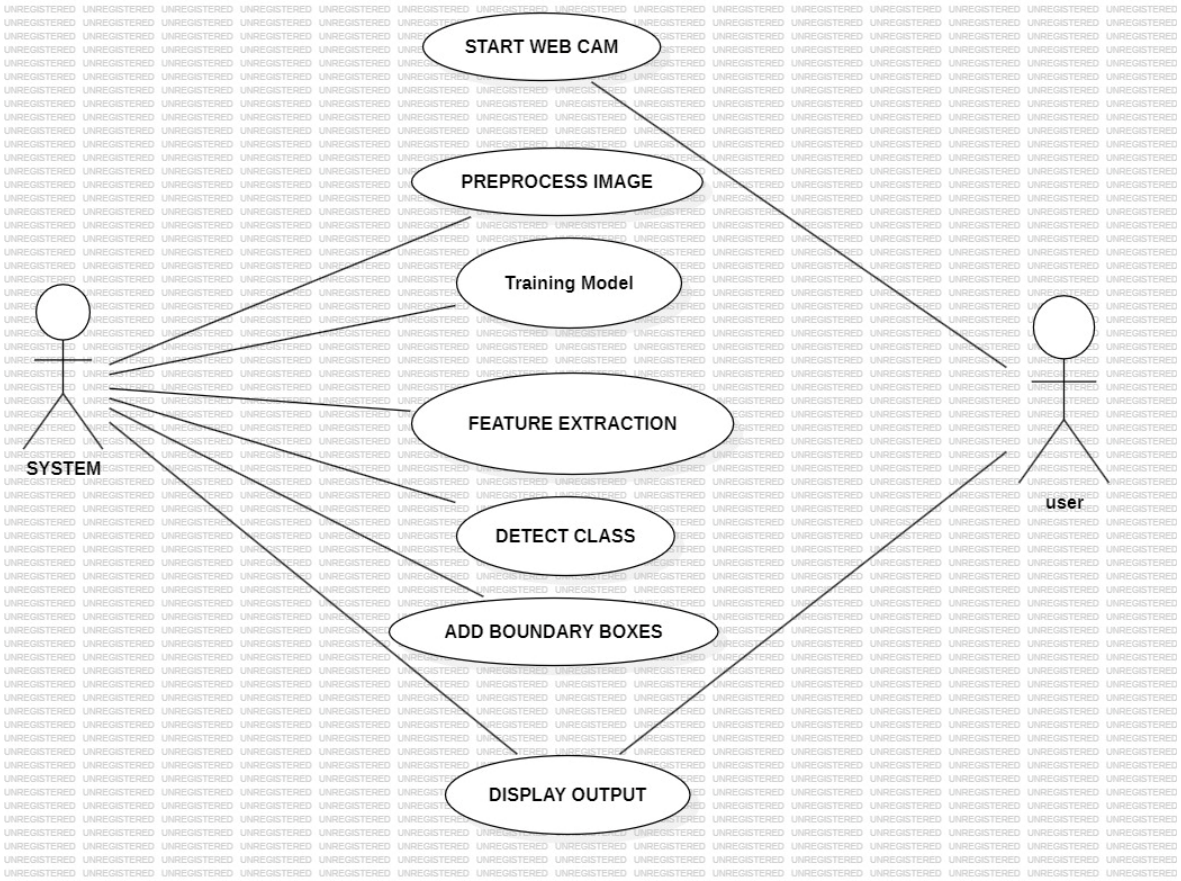

Fig-5.1 UseCase diagram

5.2 CLASS DIAGRAM:

A Class Diagram provides a visual representation of the system's structure by illustrating the classes, their attributes, methods, and relationships. The below figure provides a structured view of how different components interact within the weapon detection system. The system consists of multiple classes, each handling a specific function.

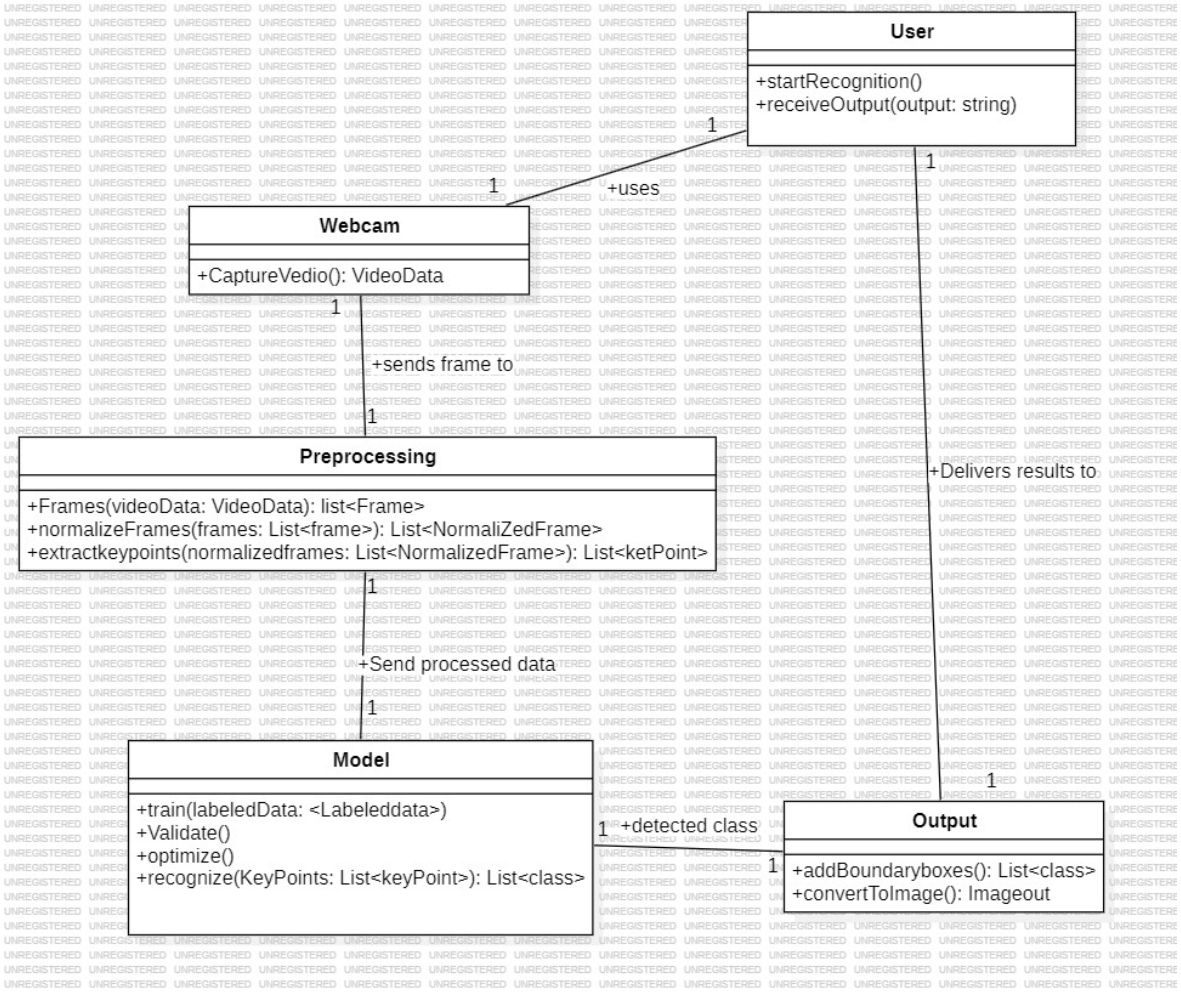

Fig-5.2 Class Diagram

- The User starts the recognition process.
- The Webcam captures video and sends frames to the Preprocessing module.
- The Preprocessing module normalizes frames and extracts key features.
- The Model processes extracted features and classifies objects as weapons or non-weapons.
- The Output module displays the detected weapon with bounding boxes.
- The User will see the result on the display.

5.3 SEQUENCE DIAGRAM:

A Sequence Diagram in Unified Modelling Language (UML) visually represents the interactions and messages exchanged between different components or objects within a system over time. Below is a simplified sequence diagram for a Weapon Detection using YOLOv11

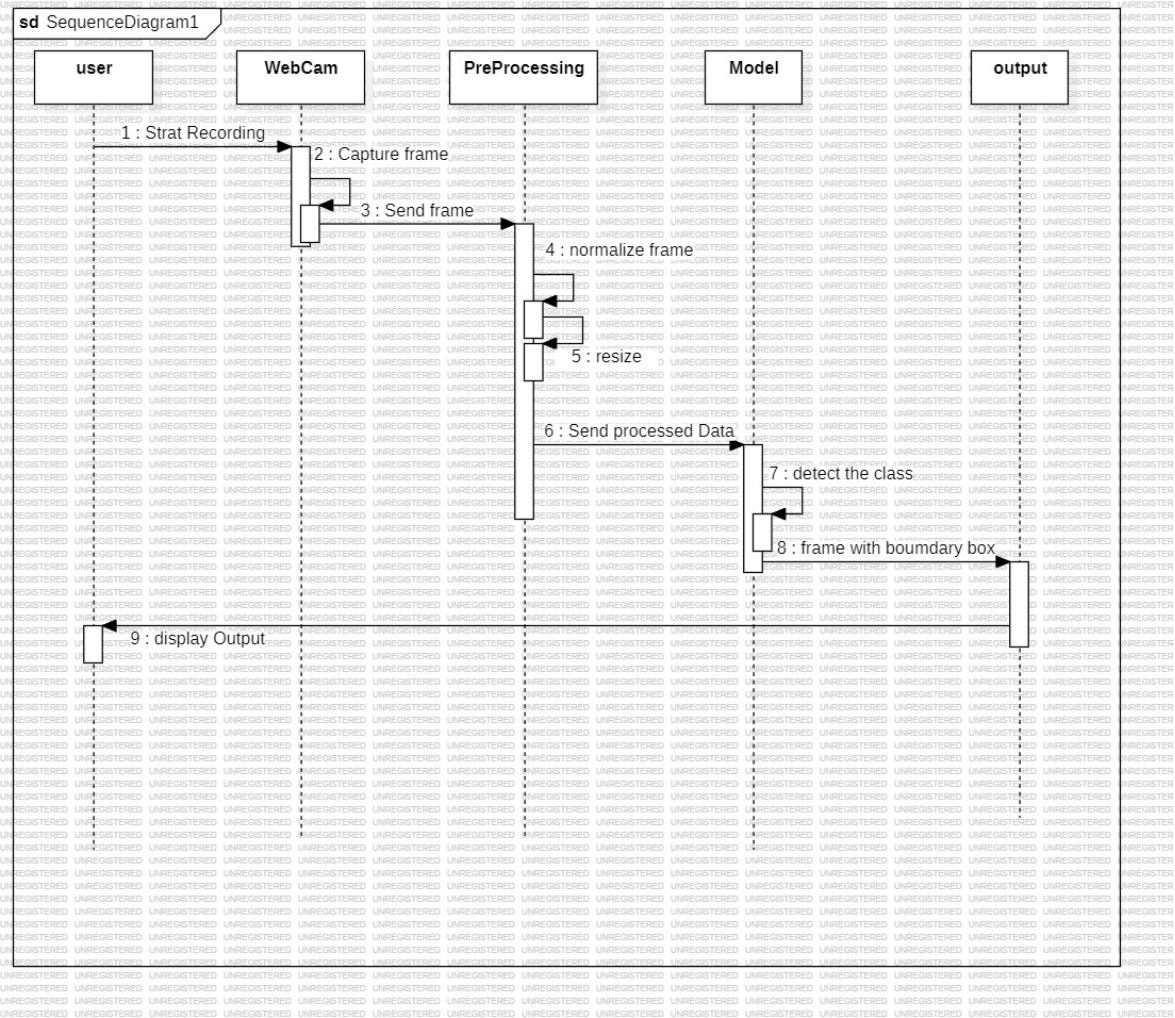

Fig-5.3 Sequence Diagram

Sequence of Operations

1. **User starts recording**: The user initiates the recording process.
2. **Webcam captures frames**: The webcam continuously captures video frames.
3. **Send frame to Preprocessing**: The captured frame is forwarded to the preprocessing module.
4. **Normalize frame**: The system processes the frame by normalizing it for better accuracy.
5. **Resize frame**: The frame is resized to a standard input size for the model.
6. **Send processed data to Model**: The processed data is sent to the ML model for classification.
7. **Detect class**: The model analyses the frame and detects whether a weapon is present.
8. **Frame with boundary box**: If a weapon is detected, the system adds a bounding box around the weapon.

9. **Display output**: The processed frame with the detected object is displayed to the user.

5.4 ACTIVITY DIAGRAM:

An Activity Diagram in Unified Modelling Language (UML) illustrates the flow of activities within a system, showing the sequence and conditions for different tasks. Below is an activity diagram representing the process flow of a Weapon Detection Using YOLOv11.

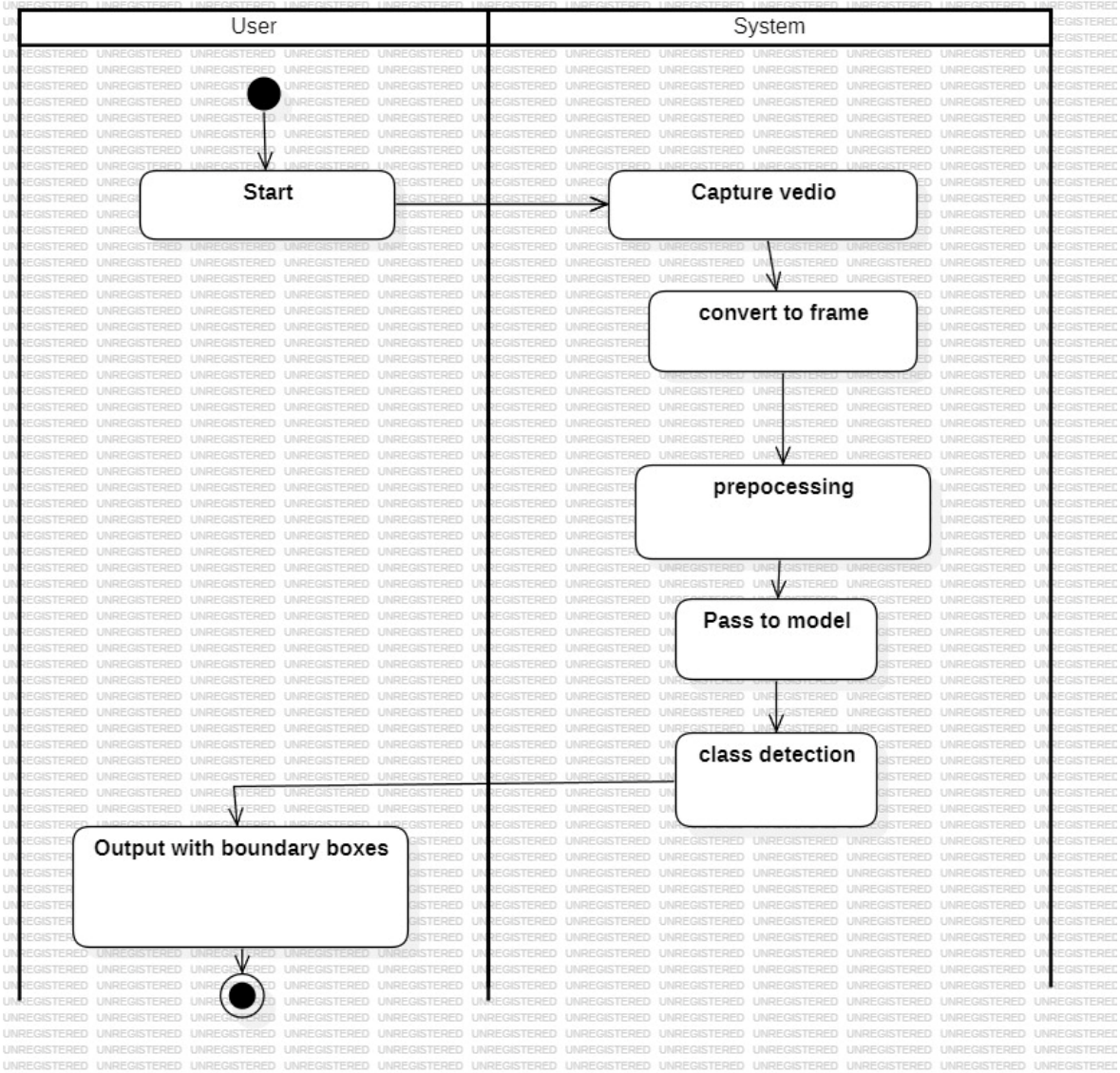

Fig-5.4 Activity Diagram

The Activity Diagram represents the step-by-step process flow of the system, focusing on how the system handles weapon detection. It consists of two main entities:

1. **User**: Initiates the process and receives the final output.
2. **System**: Handles video capture, preprocessing, model classification, and final output.

CHAPTER 6

YOLOv11 ARCHITECTURE

The architecture of YOLOv11 is designed to optimize both speed and accuracy, building on the advancements introduced in earlier YOLO versions like YOLOv8, YOLOv9, and YOLOv10. The main architectural innovations in YOLOv11 revolve around the C3K2 block, the SPFF module, and the C2PSA block, all of which enhance its ability to process spatial information while maintaining high-speed inference.

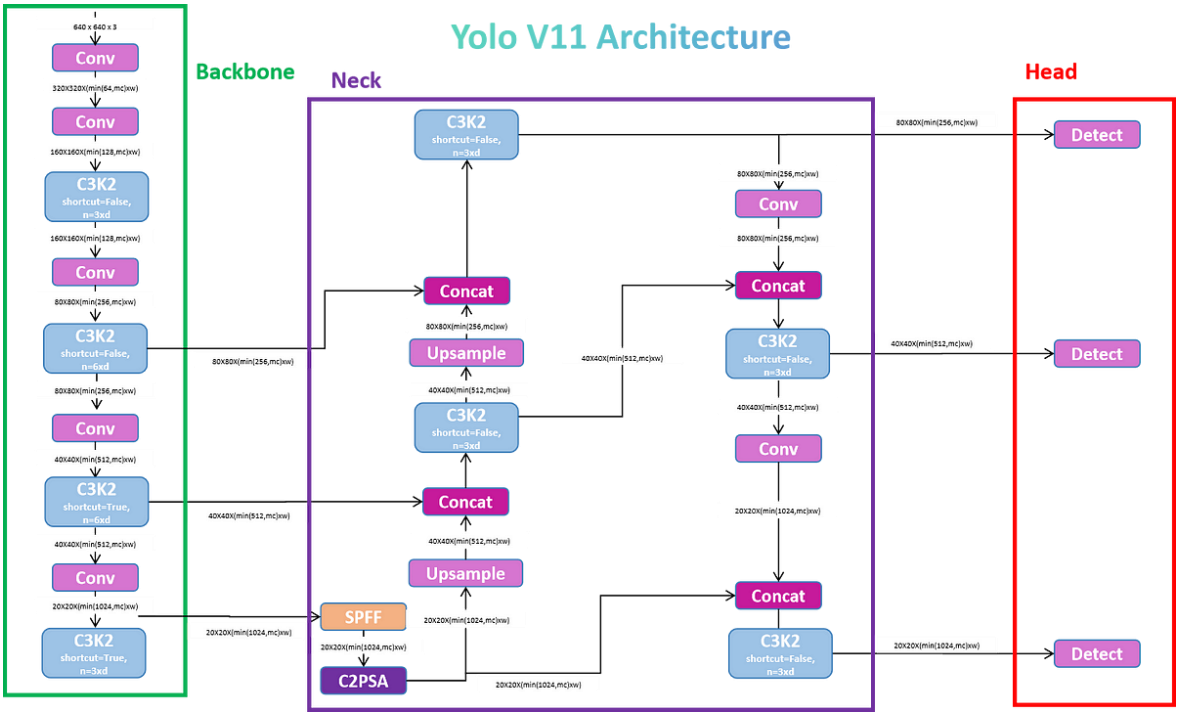

Fig-6 YOLOv11 Architecture

6.1 Backbone:

a. Convolutional Block

This block is named as Conv Block which process the given c,h,w passing through a 2D convolutional layer following with a 2D Batch Normalization layer at last with a SiLU Activation Function.

b. Bottle Neck

This is a sequence of convolutional block with a shortcut parameter, this would decide if you want to get the residual part or not. It is similar to the ResNet Block, if shortcut is set to False then no residual would be considered.

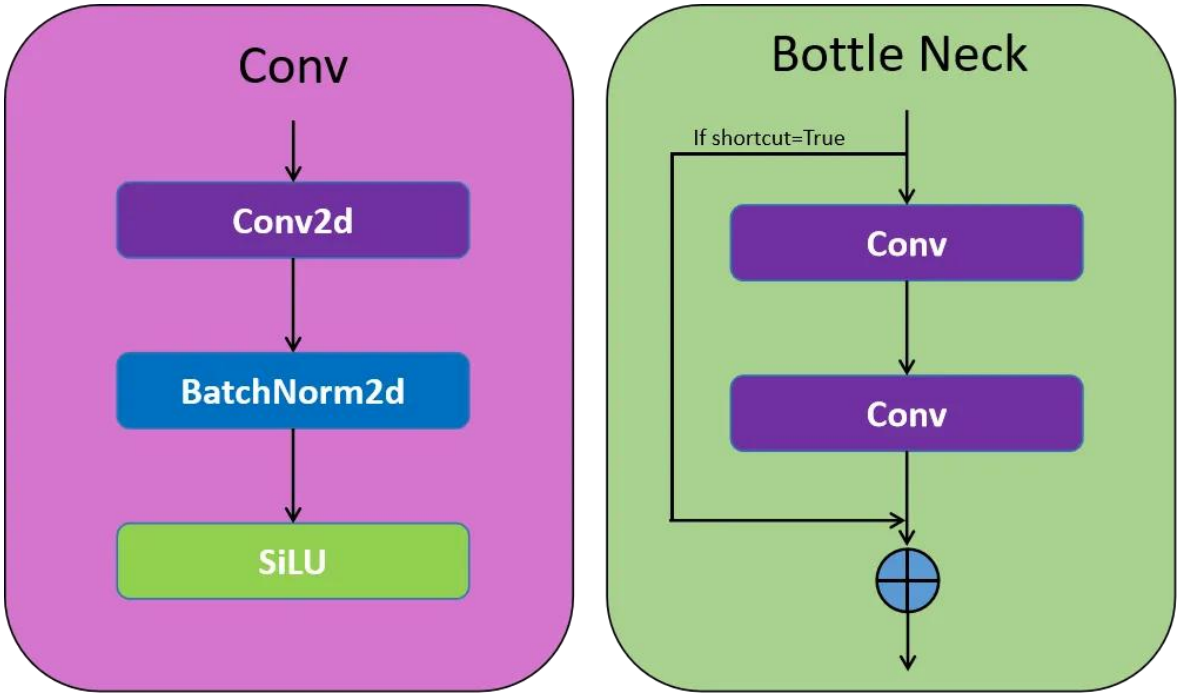

Fig-6.1.1 Convolutional Block and Bottle Neck Layer

c. C2F (YOLOv8)

The C2F block (Cross Stage Partial Focus, CSP-Focus), is derived from CSP network, specifically focusing on efficiency and feature map preservation. This block contains a Conv Block then splitting the output into two halves (where the channels gets divided), and they are processed through a series of 'n' Bottle Neck layers and lastly concatinates every layer output following with a final Conv block. This helps to enhance feature map connections without redundant information.

d. C3K2

YOLOv11 uses C3K2 blocks to handle feature extraction at different stages of the backbone. The smaller 3x3 kernels allow for more efficient computation while retaining the model's ability to capture essential features in the image. At the heart of YOLOv11's backbone is the C3K2 block, which is an evolution of the CSP (Cross Stage Partial) bottleneck introduced in earlier versions. The C3K2 block optimizes the flow of information through the network by splitting the feature map and applying a series of smaller kernel convolutions (3x3), which are faster and computationally cheaper than larger kernel convolutions. By processing smaller, separate feature maps and merging them after several convolutions, the C3K2 block improves feature representation with fewer parameters compared to YOLOv8's C2f blocks. The C3K block contains a similar structure to C2F block but no splitting will be done here, the input is passed through a Conv block following with a series of 'n' Bottle Neck layers with concatinations and ends with final Conv Block.

The C3K2 uses C3K block to process the information. It has 2 Conv block at start and end following with a series of C3K block and lastly concatinating the Conv Block output and the last C3K block output and ends with a final Conv Block. This block focuses on maintaining a balance between speed and accuracy, leveraging the CSP structure.

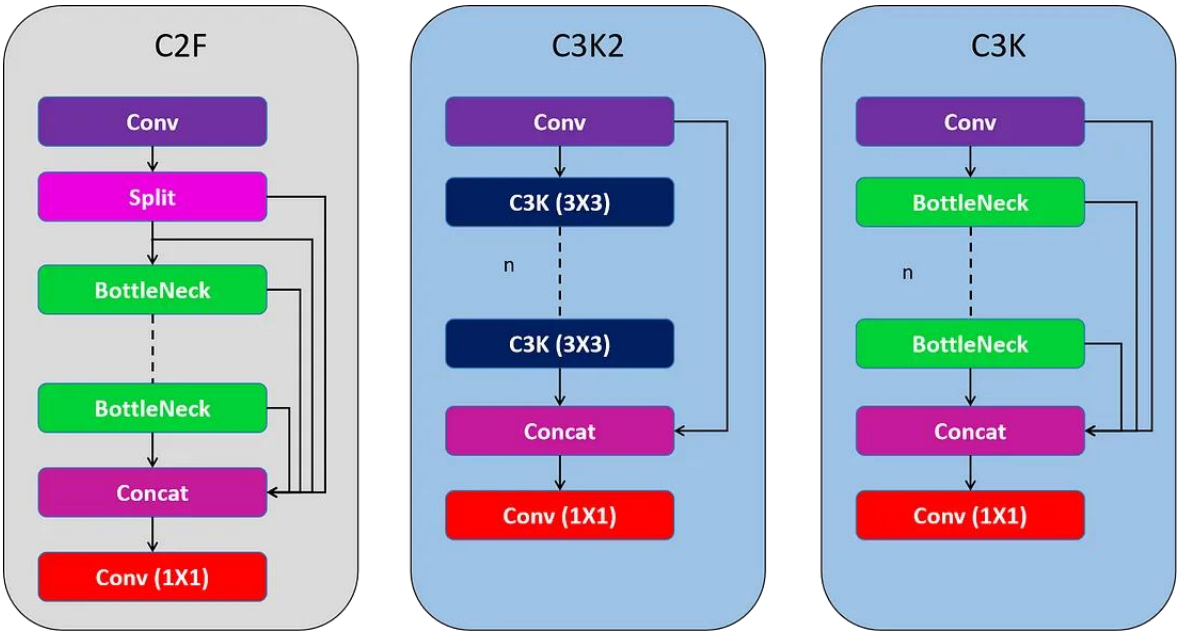

Fig-6.1.2 Comparison of C2F and C3K2 Blocks

6.2. Neck: Spatial Pyramid Pooling Fast (SPFF) and Upsampling:

YOLOv11 retains the SPFF module (Spatial Pyramid Pooling Fast), which was designed to pool features from different regions of an image at varying scales.

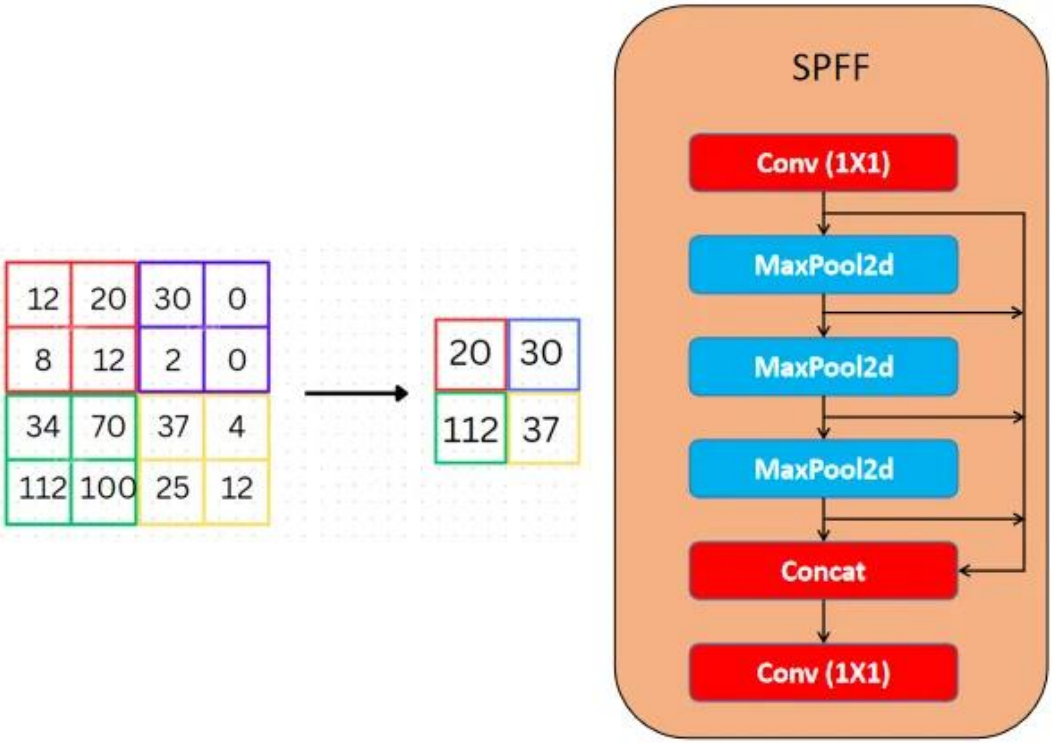

Fig-6.2 Spatial Pyramid Pooling Fast

This improves the network's ability to capture objects of different sizes, especially small objects, which has been a challenge for earlier YOLO versions.

SPFF pools features using multiple max-pooling operations (with varying kernel sizes) to aggregate multi-scale contextual information. This module ensures that even small objects are recognized by the model, as it effectively combines information across different resolutions. The inclusion of SPFF ensures that YOLOv11 can maintain real-time speed while enhancing its ability to detect objects across multiple scales.

6.3. Attention Mechanisms: C2PSA Block:

One of the significant innovations in YOLOv11 is the addition of the C2PSA block (Cross Stage Partial with Spatial Attention). This block introduces attention mechanisms that improve the model's focus on important regions within an image, such as smaller or partially occluded objects, by emphasizing spatial relevance in the feature maps.

a. Position-Sensitive Attention

This class encapsulates the functionality for applying position-sensitive attention and feed-forward networks to input tensors, enhancing feature extraction and processing capabilities. This layers includes processing the input layer with Attention layer and concatinating the input and attention layer output, then it is passed through a Feed forward Neural Networks following with Conv Block and then Conv Block without activation and then concatinating the Conv Block output and the first contact layer output.

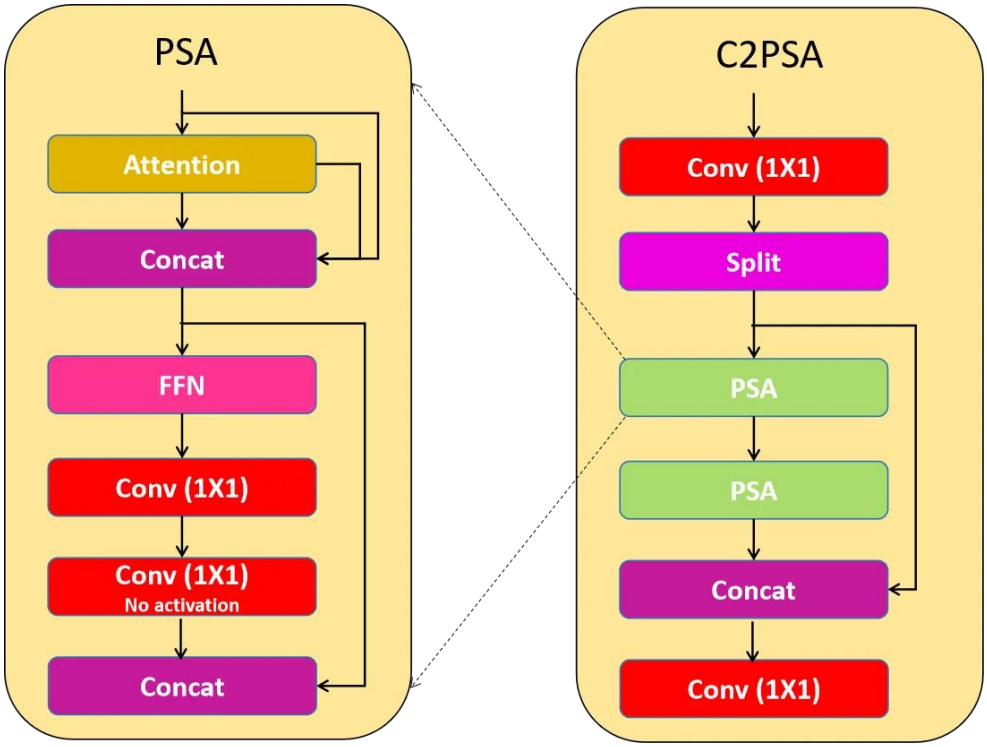

Fig-6.3 C2-Position Sensitive Attention Block (C2PSA)

b. C2PSA

The C2PSA block uses two PSA (Partial Spatial Attention) modules, which operate on separate branches of the feature map and are later concatenated, similar to the C2F block structure. This setup ensures the model focuses on spatial information while maintaining a balance between computational cost and detection accuracy. The C2PSA block refines the model's ability to selectively focus on regions of interest by applying spatial attention over the extracted features. This allows YOLOv11 to outperform previous versions like YOLOv8 in scenarios where fine object details are necessary for accurate detection.

6.4. Head: Detection and Multi-Scale Predictions

Similar to earlier YOLO versions, YOLOv11 uses a multi-scale prediction head to detect objects at different sizes. The head outputs detection boxes for three different scales (low, medium, high) using the feature maps generated by the backbone and neck.

The detection head outputs predictions from three feature maps (usually from P3, P4, and P5), corresponding to different levels of granularity in the image. This approach ensures that small objects are detected in finer detail (P3) while larger objects are captured by higher-level features (P5).

CHAPTER 7

SYSTEM DEVELOPMENT && IMPLEMENTATION

7.1 Gathering and Preparing Data:

The Weapon detection dataset is the main dataset used for the project. Bounding boxes that delineate each weapon in an image and provide the coordinates of its location are commonly included in the dataset. This aids the model's learning of the weapon's location within the frame. For multi-class classification jobs, it is essential that each bounding box has a label identifying the type of weapon (e.g., knife, gun, rifle, etc.).

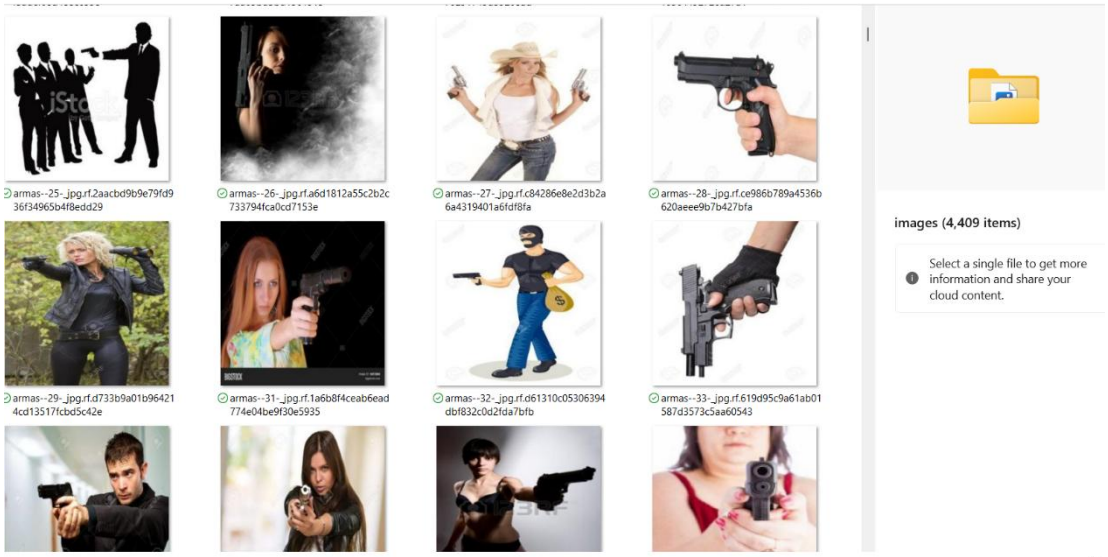

Fig-7.1.1 Dataset of weapons

```
0 0.185 0.6833333333333333 0.058333333333333334 0.06
```

Fig-7.1.2 Label of an image

Characteristics of the Dataset: There are 4,409 images in the dataset. The YOLOv11 format is used to annotate gun detection. We preprocessed the dataset by leveling brightness and image sizes to improve consistency and guarantee the best possible model performance. In order to increase the model's resilience to real-world circumstances, we also used data augmentation approaches to add variability, building and training, which allowed our system to detect weapons in a variety of scenarios with high accuracy.

7.2 Augmentation of Data: To improve the dataset and the generalization of the model, a variety of augmentation approaches were used, such as random rotations, flips, brightness modifications, and scaling. This stage was especially helpful in guaranteeing the model's resilience in various crowd densities and illumination scenarios. We used occlusion augmentation, which partially obscures objects in some training photos, to better mimic real-world settings. This aids the model's ability to identify firearms in congested settings where people might block one another.

7.3Install and import dependences:

To implement a Weapon Detection system, import and install the necessary dependencies using Python and a virtual environment. Utilize the following commands:

```
1  !pip install ultralytics
2  from IPython.display import display , Image
3  import os
4  from IPython import display
5  import numpy as np
6  import wandb
7  from ultralytics import YOLO
8  import kagglehub
9  import matplotlib.pyplot as plt
10 import matplotlib.image as mpimg
11 import numpy as np
```

Table-1 Installing and Importing Dependences

7.4Import Dataset:

Dataset is most important in the model accuracy and efficiency. The dataset with large number of images will increase the model accuracy. The dataset is imported from the Kaggle hub the dataset contains the weapons(knife and gun) images with label of each image.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("iqmansingh/guns-knives-object-detection")

print("Path to dataset files:", path)
print("Dataset files:", os.listdir(path))
```

Table-2 Importing Datsets

7.5Model Training:

For training a Weapon Detection system, typically used a YOLOv11 Model.

Model YOLOv11n. for 100 epochs and input the images size of (640,640) images.

```
!yolo task=detect mode=train data='/root/.cache/kagglehub/datasets/iqmansingh/guns-knives-object-detection/versions/5/guns-knives-yolo/guns-knives-yolo/data.yaml'  model="yolo11n.pt" epochs=100 imgsz=640
```

Table-3 Yolo Training

7.6Model Accuracy:

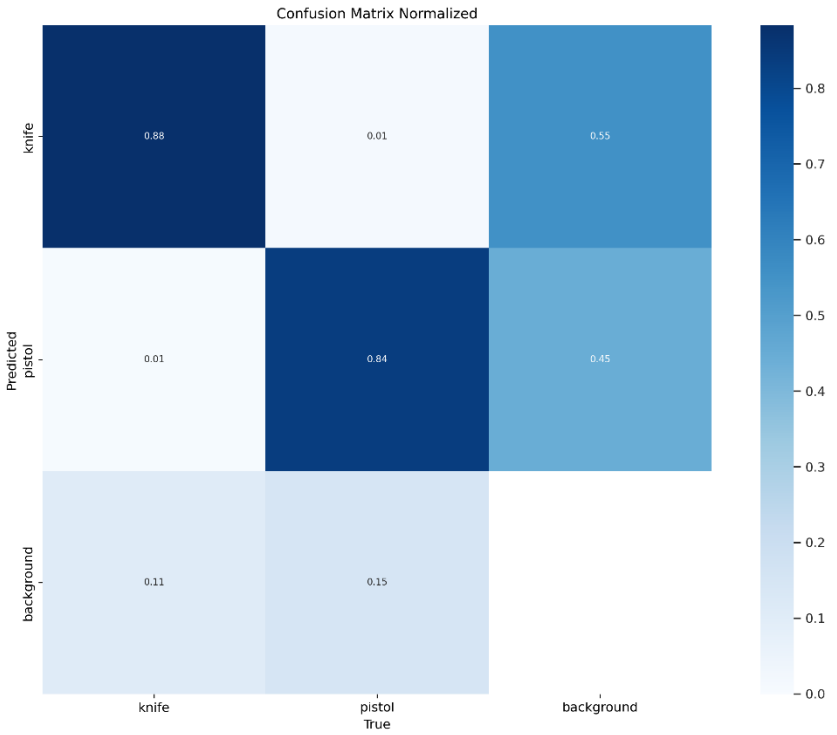

Fig-7.6 Confusion Matrix

- A confusion matrix helps us evaluate the performance of a classification model.

- The model correctly predicts "knife" 89% of the time but misclassifies 11% as "background".
- The model correctly predicts "pistol" 89% of the time but misclassifies 41% as "background".
- The background class is confused with other classes, meaning some images that should be classified as background are misidentified as weapons.
- The model has strong accuracy for weapon detection (knife & pistol) but struggles to differentiate background from weapons.

7.7Model Testing:

After training the model on the custom dataset of weapons, evaluate its performance using testing data. Employ metrics such as accuracy, precision, recall, and F1 score.

```
1  !yolo task=detect mode=val model='/content/runs/detect/train/weights/best.pt' data =
   '/root/.cache/kagglehub/datasets/iqmansingh/guns-knives-object-detection/versions/5/guns-knives-yolo/guns-knives-yolo/data.yaml'
2
3  Ultralytics 8.3.79 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
4  YOLO11n summary (fused): 100 layers, 2,582,542 parameters, 0 gradients, 6.3 GFLOPs
5  val: Scanning /root/.cache/kagglehub/datasets/iqmansingh/guns-knives-object-detection/versions/5/guns-knives-yolo/guns-knives-yolo/valid/labels.cache... 1043 images, 7 backgrounds, 0 corrupt: 100% 1043/1043 [00:00<?, ?it/s]
6           Class  Images  Instances      Box(P       R   mAP50  mAP50-95): 100% 66/66 [00:11<00:00,  5.88it/s]
7             all    1043       1203      0.913      0.849      0.921      0.689
8            knife     542        601      0.896      0.842      0.915      0.638
9           pistol     494        602      0.931      0.856      0.927      0.741
10 Speed: 0.6ms preprocess, 3.9ms inference, 0.0ms loss, 1.4ms postprocess per image
11 Results saved to runs/detect/val
```

Table-4 Model Testing

Trained YOLOv11 model has been evaluated on the guns and knives object detection dataset. Below is a breakdown of the results:

- **High Precision (0.91 overall)**: The model correctly identifies weapons with minimal false positives.
- **Good Recall (0.84 overall)**: It detects most of the actual weapons in images.
- **mAP@50 (0.92)**: Strong performance in detecting both knives and pistols.

CHAPTER 8

DEPLOYMENT CODE

8.1 Flask Code for model deployment

- app.py-filename

Table-5 Deployment Using Flask

```
import cv2
import os
import time
import pyautogui
import pywhatkit as kit
from flask import Flask, Response, render_template, jsonify
from ultralytics import YOLO
from threading import Lock, Thread
from deep_sort_realtime.deepsort_tracker import DeepSort
import random
from queue import Queue

app = Flask(__name__)

model = YOLO("C:/Users/prave/OneDrive/Desktop/weapon_detection/best2.pt")

camera = None
camera_running = False
lock = Lock()

SAVE_FOLDER = "detected_frames"
os.makedirs(SAVE_FOLDER, exist_ok=True)

PHONE_NUMBER = "+9163019469xx"

tracker = DeepSort(max_age=15)

detected_objects = {}  # Dictionary to store saved frames for each track ID
color_map = {}
object_classes = {}  # Store class name for each tracked object ID

message_queue = Queue()


def get_random_color():
    return tuple(random.randint(100, 255) for _ in range(3))


def send_whatsapp_messages():
    while True:
        image_path, weapon_count, detected_classes = message_queue.get()
        if image_path:
            caption = f"🚨 Weapon Detected!\n🛑 Count: {weapon_count}\n🔹 Classes: {', '.join(detected_classes)}"
            time.sleep(2)
            kit.sendwhats_image(phone_no=PHONE_NUMBER, img_path=image_path, caption=caption)
            print("Waiting for WhatsApp Web to load...")
            time.sleep(3)
            pyautogui.press('enter')
            print(f"WhatsApp message sent with image: {image_path}")
        message_queue.task_done()

Thread(target=send_whatsapp_messages, daemon=True).start()


def generate_frames():
    global camera, camera_running, detected_objects, object_classes
```

```
    while True:
        with lock:
            if not camera_running or camera is None:
                break
            success, frame = camera.read()
            if not success:
                break

            results = model(frame, conf=0.5)
            detections = []

            for result in results:
                for box in result.boxes:
                    x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
                    conf = box.conf[0].item()
                    cls = int(box.cls[0].item())
                    if conf > 0.5:
                        detections.append(([x1, y1, x2 - x1, y2 - y1], conf, cls))

            tracked_objects = tracker.update_tracks(detections, frame=frame)

            for obj in tracked_objects:
                if not obj.is_confirmed():
                    continue

                track_id = obj.track_id
                ltrb = obj.to_ltrb()
                x1, y1, x2, y2 = map(int, ltrb)

                if track_id not in object_classes:
                    for det in detections:
                        if abs(x1 - det[0][0]) < 10 and abs(y1 - det[0][1]) < 10:
                            object_classes[track_id] = model.names[det[2]]
                            break

                class_name = object_classes.get(track_id, None)
                if class_name is None:
                    continue

                if track_id not in color_map:
                    color_map[track_id] = get_random_color()
                color = color_map[track_id]

                if track_id not in detected_objects:
                    detected_objects[track_id] = True
                    timestamp = int(time.time())
                    image_path = os.path.join(SAVE_FOLDER, f"{class_name}_ID{track_id}_{timestamp}.jpg")

                    frame_copy = frame.copy()
                    cv2.rectangle(frame_copy, (x1, y1), (x2, y2), color, 2)
                    cv2.putText(frame_copy, f"{class_name} ID: {track_id}", (x1, y1 - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)
                    cv2.imwrite(image_path, frame_copy)
                    message_queue.put((image_path, 1, [class_name]))
                    print(f"Frame saved: {image_path}")

                cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
                cv2.putText(frame, f"{class_name} ID: {track_id}", (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

            _, buffer = cv2.imencode('.jpg', frame)
            frame_bytes = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')


@app.route('/')
```

```
def index():
    return render_template('ind.html')


@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')


@app.route('/start_camera', methods=['GET'])
def start_camera():
    global camera, camera_running, detected_objects, object_classes
    with lock:
        if not camera_running:
            camera = cv2.VideoCapture(0)
            if not camera.isOpened():
                return jsonify({"status": "error", "message": "Failed to access camera"}), 500
            camera_running = True
            detected_objects = {}
            object_classes = {}
            return jsonify({"status": "started"})
    return jsonify({"status": "already running"})


@app.route('/stop_camera', methods=['GET'])
def stop_camera():
    global camera, camera_running
    with lock:
        if camera_running:
            camera_running = False
            if camera:
                camera.release()
                camera = None
            return jsonify({"status": "stopped"})
    return jsonify({"status": "already stopped"})


@app.route('/get_saved_image', methods=['GET'])
def get_saved_image():
    if detected_objects:
        return jsonify({"message": "Images saved for detected objects."})
    return jsonify({"message": "No image detected yet"}), 404


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

8.2 Website Code:

- index.html-filename

Table-6 User Interface

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Live Weapon Detection</title>
    <style>
        body {
            background-color:rgb(236, 248, 7);
            color: white;
```

```
            text-align: center;
            font-family: Arial, sans-serif;
        }
        h1 {
            margin-top: 20px;
            color:black;
        }
        button {
            padding: 10px 20px;
            font-size: 18px;
            margin: 10px;
            border: none;
            cursor: pointer;
            border-radius: 25px;
            border-style: solid;
            border-color:whilte;
        }
        #start {
            background-color: green;
            color: white;
        }
        #stop {
            background-color: red;
            color: white;

        }
        #video_feed {
            width: 60%;
            margin-top: 20px;
            border: 3px solid white;
            display: block;
            margin-left: auto;
            margin-right: auto;
        }
    </style>
</head>
<body>
    <h1>Live Weapon Detection</h1>
    <button id="start" onclick="startCamera()">Start Camera</button>
    <button id="stop" onclick="stopCamera()">Stop Camera</button>

    <div>
        <img id="video_feed" src="" alt="Live Video Feed">
    </div>

    <script>
        function startCamera() {
            fetch('/start_camera')
                .then(response => response.json())
                .then(data => {
                    if (data.status === "started") {
                        document.getElementById("video_feed").src = "/video_feed?" + new
Date().getTime();
                    } else if (data.status === "error") {
```

```
                    alert("Failed to access camera. Please check camera permissions.");
                }
            })
            .catch(error => console.error("Error:", error));
        }

        function stopCamera() {
            fetch('/stop_camera')
            .then(response => response.json())
            .then(data => {
                if (data.status === "stopped") {
                    document.getElementById("video_feed").src = "";
                }
            })
            .catch(error => console.error("Error:", error));
        }
    </script>
</body>
</html>
```

CHAPTER 9

OUTPUT SCREENS

9.1 Run flask file:

```
PS C:\Users\prave\OneDrive\Desktop\weapon_detection> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.245.64:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
```

9.2 Website:

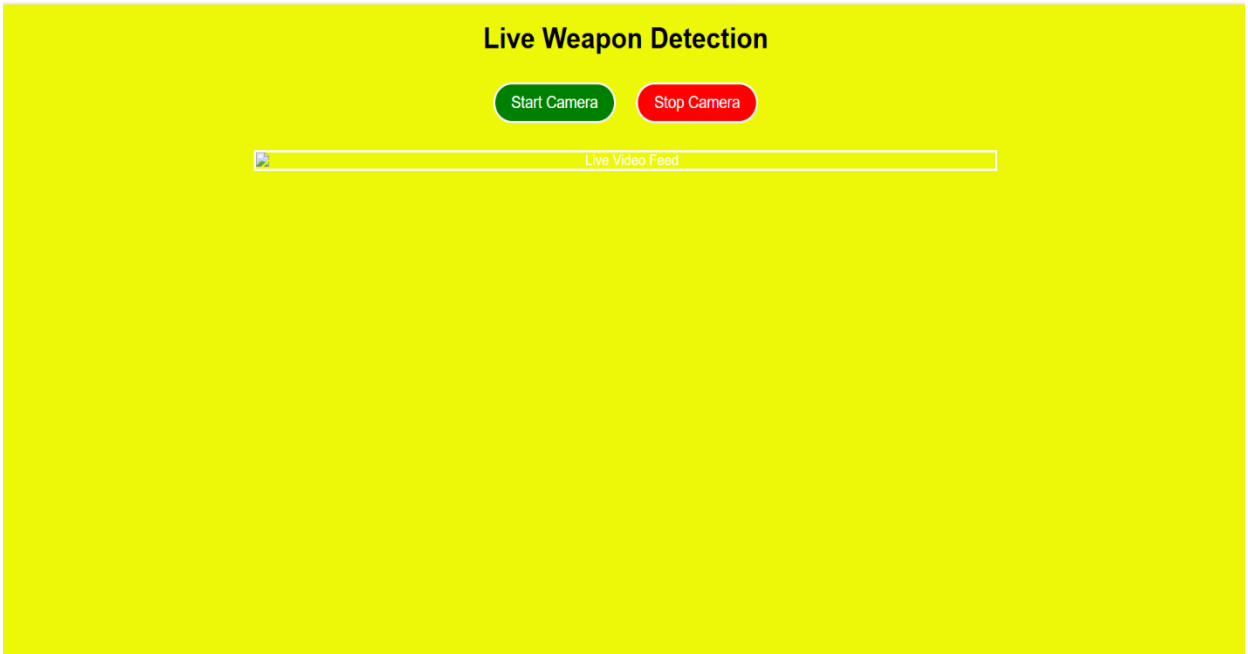

Fig-9.2 Website

9.3 Before camera start:

Fig 9.4 Before camera start

9.4 Live Detecting:

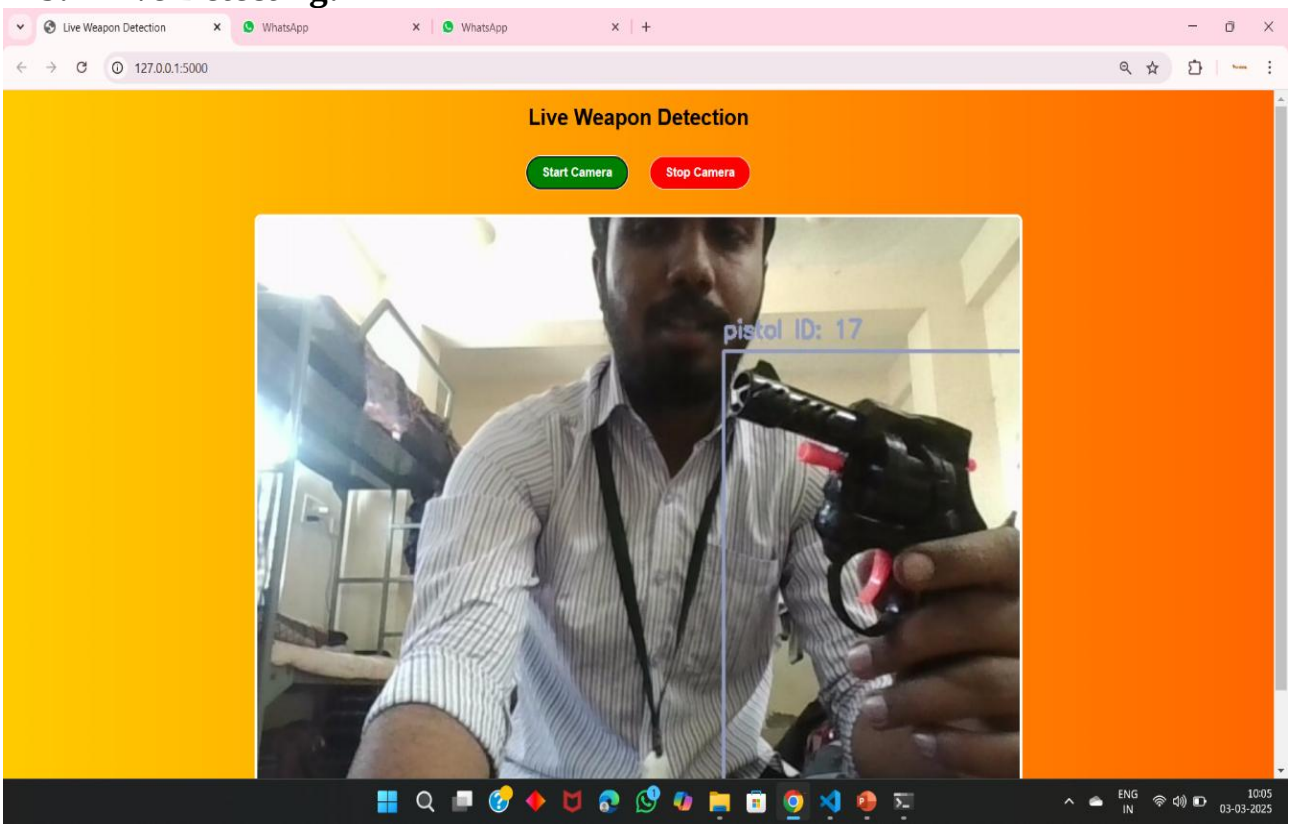

Fig-9.4 Website while live detecting

9.5 WhatsApp alert message:

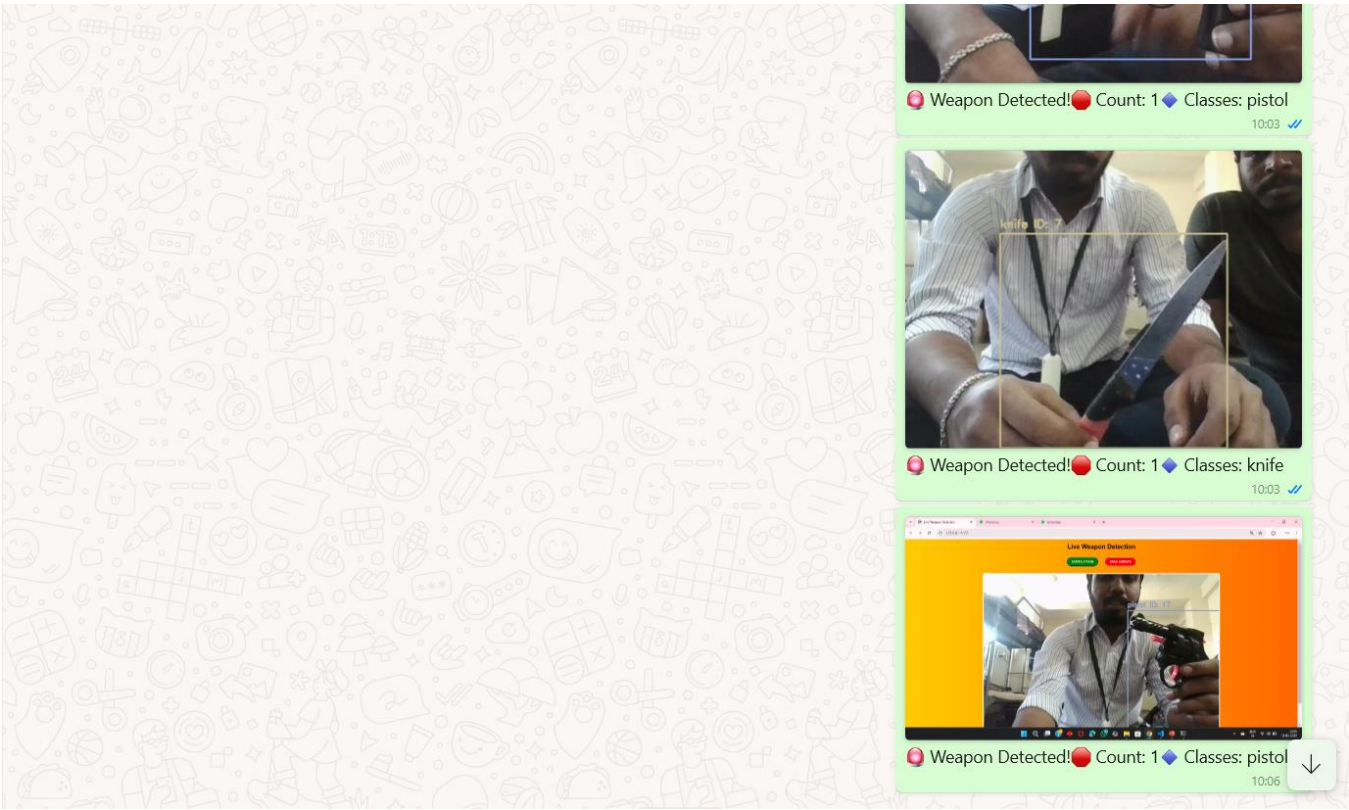

Fig-9.5 WhatsApp alert message

CHAPTER 10

CONCLUSION AND FUTURE SCOPE

In conclusion, Significant developments in AI-driven surveillance systems, especially those intended for effective crime prevention. The technology greatly improves real-time surveillance by automating the detection of suspicious behaviors and weapon detection through the integration of YOLOv11 object detection algorithms and machine learning models, thereby lowering the need for human involvement by 60–70%. As a result, possible dangers are addressed more quickly and precisely. The system's remarkable accuracy rating of 96.3% in activities like weapon surveillance and weapon identification shows that it can manage intricate security issues in busy public spaces. By lowering latency by 30–40%, edge computing improves system performance even more and enables real-time data processing and more effective decision-making. When taken as a whole, these enhancements result in improved situational awareness and more proactive crime prevention strategies. Our system's scalability allows it to be tailored to different public areas, offering a reliable public safety management solution. To further improve the system's impact on public safety, future research will try to maximize computing efficiency, investigate new applications, and extend the system's deployment across various environments.

FUTURESCOPE

● **Improved Detection Capabilities:** The system's use in critical infrastructure monitoring and public safety can be expanded by training it with a wider range of datasets to identify additional threats such as fire, abandoned bags, or hazardous materials.

● **Integration with Smart Cities**: It can be used into smart city solutions to improve public safety and emergency response in urban settings by automating real-time crowd monitoring and threat detection.

CHAPTER 11

REFRENCES

[1] Dudu Guo, Zhuzhou Li, Hongbo Shuai, and Fei Zhou. Multi-target vehi cle tracking algorithm based on improved deepsort. Sensors, 24(21):7014, 2024.

[2] H. K. Saini, Kussum and Mandeep, "Artificial Intelligence and Internet of Things: A Boon for the Crime Prevention," 2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT), 2023,pp.7-11,doi:10.1109/ICAICCIT60255.2023.10466124 Faridabad, India,

[3] H. A. AlSuwaidi, S. Harous and M. A. Serhani, "Review of visual AI applications in smart surveillance for crime detection and prevention: a survey," 4th International Conference on Distributed Sensing and Intelligent Systems (ICDSIS 2023), Dubai, UAE, 2023, pp. 307-320, doi: 10.1049/icp.2024.0502.

[4] Nidhal Jegham, Chan Young Koh, Marwan Abdelatti, and Abdeltawab Hendawi. Evaluating the evolution of yolo (you only look once) models: A comprehensive benchmark study of yolo11 and its predecessors. arXiv preprint arXiv:2411.00201, 2024.

[5] G. Abdiyeva-Aliyeva, M. Hematyar and S. Bakan, "Development of System for Detection and Prevention of Cyber Attacks Using Artificial Intelligence Methods," 2021 2nd Global Conference 10.1109/GCAT52182.2021.9587584..

[6] Telugu Maddileti, J Sirisha, Rayudu Srinivas, K Saikumar, et al. Pseudo trained yolo r cnn model for weapon detection with a real-time kaggle dataset. International Journal of Integrated Engineering, 14(7):131–145, 2022.

[7] P. Shrivastav and V. R. J, "A Real-Time Crowd Detection and Monitoring System using Machine Learning," 2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), Bengaluru, India, 2023, pp. 383-388, doi: 10.1109/IDCIoT56793.2023.10053517.

[8] Robert Susmaga. Confusion matrix visualization. In Intelligent Informa tion Processing and Web Mining: Proceedings of the International IIS: IIPWM ‘04 Conference held in Zakopane, Poland, May 17–20, 2004, pages 107–116. Springer, 2004

[9] C. S. Manikandababu, J. M, K. B and N. S, "Smart Crowd Monitoring System using Image Processing Technique During Covid-19," 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy,

10.1109/ICOSEC54921.2022.9951874. India, 2022, pp. 1472-1475, doi:

[10] M. H. K. Khel et al., "Realtime Crowd Monitoring—Estimating Count, Speed and Direction of People Using Hybridized YOLOv4," in IEEE Access, vol. 11, pp. 56368-56379, 2023, doi: 10.1109/ACCESS.2023.3272481.

[11] M. Abdou and A. Erradi, "Crowd Counting: A Survey of Machine Learning Approaches," 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2020, pp. 48-54, doi: 10.1109/ICIoT48696.2020.9089594.