Project Report

on

“Real-time threat detection using YOLO v11for enhanced surveillance”

Submitted by

Aryan Dani
Sobaan Jagirdar
Prakhar Jaiswal
P. Swayamprakash

Under the guidance of
Prof. Jyoti G. Mante (Khurpade)

In partial fulfilment of

Diploma in Computer Engineering

[2024-2025]

At

DEPARTMENT OF COMPUTER ENGINEERING

MIT POLYTECHNIC PUNE-411038

Affiliated to

Acknowledgments

I am profoundly grateful to **Prof. Jyoti G. Mante (Khurpade)** HOD Computer Engineering Department for her expert guidance and continuous encouragement throughout to see that this project meets its target since its commencement to its completion.

I would like to express my deepest appreciation towards **Prof. R. S. Kale** Principal, MAEER'S MIT POLYTECHNIC, Pune, whose invaluable guidance supported me in completing this project. Finally, I must express my sincere heartfelt gratitude to all the staff members of the Computer Engineering Department who helped me directly or indirectly during this course of work.

NAME OF STUDENTS

Mr. Aryan Dani

Mr. Prakhar Jaiswal

Mr. Sobaan Jagirdar

Mr. Swayamprakash Patro

DIPLOMA IN COMPUTER ENGINEERING

ABSTRACT

In today's world, the call for surveillance systems is high, but there are many problems with our current surveillance systems. It becomes challenging due to the inability of humans to closely monitor the videos from surveillance systems. There is a dire need for an automated weapon/threat detection system that can minimize these types of occurrences. The system we built uses YOLOv11s, a well-trained model that can be highly accurate for object detection in cases of weapons or threats. This study mainly revolves around the concept of machine learning and image classification as a whole using bounding boxes to identify multiple weapon/threat objects within a specific video frame in real time. The system was trained and evaluated on a massive dataset of knives, pistols, guns and other firearms demonstrating an impressive accuracy of XX%. Recently, deep learning models such as YOLO, SSD and Faster R-CNN have taken away much of the hype because of their remarkable performance. We evaluated the model's performance using key metrics such as precision, recall, F1-score, and mean Average Precision (mAP) across multiple Intersection over Union (IoU) thresholds, revealing a significant capability to differentiate between weapon and non-weapon classes with minimal error. Furthermore,we assessed the system's operational efficiency, demonstrating that it can process frames at high speeds suitable for real-time applications.

Keywords: Threat Detection, Automated Surveillance, Yolov11, Threat Detection, Computer Vision, Machine Learning, Deep Learning, Image Classification, Firearm Identification, Public Safety

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Introduction

1.1 Motivation:

The incentive of forging this project is to automate and improve the entire tedious process of traditional surveillance systems that we see today. As a team we observed how just setting this system up initially can save so many lives but at the same time it reallocates human efforts towards more strategic and meaningful tasks, as the burden of constant monitoring is removed. All key problems that we had with traditional systems can be resolved using this This project aims to revolutionize surveillance with intelligent, automated, and highly reliable threat detection.

1.2 Background:

Delays in danger detection, manual monitoring, and human error are some of the drawbacks of the conventional surveillance systems used in public areas. It is frequently ineffective to see suspicious activity or unsecured items in a crowd, which might be dangerous for security. Scalability issues and dynamic environment adaptation plague current systems. In order to solve these issues, this project introduces an AI-powered surveillance system that makes use of the YOLO v11 deep learning model. Timely notifications, automatic monitoring, and real-time danger identification have all been made possible by this technology. With the least amount of human involvement, the system claims to improve security, lower operational faults, and guarantee efficient observation.

1.3 Need of the Project:

With increasing terrorist and criminal activities around the world, detecting several types of weapons at the same time has become crucial. Most of these groups use lightweight weapons which are easily lifted and run with them. In public places like Airports, Railway Stations, Malls etc where most incidents happen, it is nearly impossible to rely on traditional surveillance systems where humans are involved to sit behind a monitor to detect a threat. For that reason, developing an intelligent system capable of discovering these types of weapons as threats beforehand with high accuracy and speed becomes of utmost necessity.

1.4 Introduction of Project:

According to statistics, there is an increase in crime during crowded events; hence, security is always a top priority in all fields. Weapons such as pistols and knives are used by criminals to commit crimes, arson, assaults, arrests, road accidents, explosions, burglary, fighting, abuse, robbery, shooting, shoplifting, stealing, vandalism, etc. The system is developed using YOLO v11 and CNN models for achieving high accuracy with minimal latency in crowded environments where quick decisions are needed.

The focus of this project will be to develop an automated threat detection system using deep learning models, optimized with edge and cloud-based processing for surveillance. The proposed system is based on CNN and YOLOv8 models, which have high accuracy and low latency in crowded environments where a fast decision is required. In the past, hardware limitations in processing power caused problems for automated real-time threat detection. Optimized deep learning models provide a good balance between performance and efficiency and also make detection on edge devices feasible.

Human detection in dynamic environments becomes even more challenging due to variations in lighting, occlusions, and diverse human postures while they hold an object that poses a threat. In this line, the project implements advanced image processing techniques and deep learning models for high-speed and accurate detection.

The system is designed with a web-based user interface for real-time visualization of detection and segmentation. It will also support functionalities like live camera feeds and image uploads, which makes it ideal for surveillance applications where quick response and accurate threat detection are needed.

A Comparative Analysis of Weapons Detection Using Various Deep Learning Techniques [1]

The paper, "A Comparative Analysis of Weapons Detection Using Various Deep Learning Techniques," explores the application of state-of-the-art deep learning methods for weapons detection, providing a comprehensive evaluation of their performance in diverse scenarios. The study investigates popular object detection frameworks, including Convolutional Neural Networks (CNNs), YOLO (You Only Look Once), Faster R-CNN (Region-Based Convolutional Neural Network), SSD (Single Shot MultiBox Detector), and RetinaNet, emphasizing their strengths and weaknesses in detecting weapons under varying conditions. These models are trained and tested on publicly available datasets like COCO (Common Objects in Context), PASCAL VOC, and domain-specific datasets such as RWD (Real-World Weapon Dataset), which feature a mix of images and video frames containing firearms, knives, and other potential threat objects. Additionally, synthetic datasets generated through data augmentation techniques are used to evaluate the robustness of the models in handling edge cases, such as partial occlusion, low-light conditions, and crowded scenes. The evaluation metrics include precision, recall, F1-score, and mean Average Precision (mAP), along with the computational efficiency and inference speed of the models, making the analysis holistic. The results highlight that YOLOv4 and YOLOv5 models excel in real-time applications due to their balance of accuracy and speed, while Faster R-CNN and RetinaNet demonstrate superior detection rates in scenarios requiring high precision, such as identifying concealed weapons. SSD, with its lightweight architecture, is shown to be particularly effective for resource-constrained environments like mobile devices. Furthermore, the study emphasizes the importance of dataset diversity and quality in achieving reliable detection results, recommending further research to create larger, more varied datasets for training. By comparing these models, the paper provides valuable insights for enhancing intelligent surveillance systems, which can be applied to critical domains like airport security, public event monitoring, and urban policing, ensuring improved threat detection and public safety measures. This work highlights the need for tailoring the choice of the detection model to the specific operational requirements and environmental constraints, which should foster advancements in automated security technologies.

Interpretable Features of YOLO v8 for Weapon Detection - Performance Driven Approach [2]

The paper, "Interpretable Features of YOLO v8 for Weapon Detection - Performance Driven Approach," explores the implementation of the YOLO v8 (You Only Look Once version 8) deep learning model for detecting weapons in real-time scenarios, with a focus on improving the interpretability of its features to enhance detection performance and reliability. YOLO v8, as the most stable iteration in the YOLO family, offers significant advancements in terms of accuracy and speed, making it well-suited for security applications requiring rapid threat detection. The study utilizes diverse datasets that include images and videos containing various weapon types such as firearms, knives, and other sharp objects. Among the datasets employed are the widely recognized COCO (Common Objects in Context) and Open Images Dataset, supplemented by domain-specific datasets curated for weapons detection, ensuring a mix of real-world and synthetic scenarios. Additionally, the authors emphasize the use of data augmentation techniques like rotation, scaling, and contrast adjustment to simulate challenging conditions such as occlusion, low-light environments, and motion blur.

The research highlights the interpretability of YOLO v8's features as a critical factor in understanding its decision-making process. By employing techniques like Grad-CAM (Gradient-weighted Class Activation Mapping) and Feature Visualization, the authors examine how the model distinguishes weapons from other objects in complex environments. This interpretability not only aids in improving detection accuracy but also builds trust in the model's predictions, which is vital for high-stakes applications like public safety and law enforcement. YOLO v8's architecture, with its advanced anchor-free detection mechanism and enhanced convolutional layers, is specifically optimized to handle a variety of object sizes and orientations, ensuring consistent performance across diverse scenarios.

The paper uses evaluation metrics such as precision, recall, F1-score, and mean Average Precision (mAP) to completely estimate the effectiveness of the model. The results show that YOLO v8 performs better than its predecessors in both accuracy and processing speed; hence, it is quite suitable for real-time surveillance systems in airports, stadiums, and urban monitoring systems. Moreover, areas of further improvement are already pointed out, such as the enhancement of the ability of the model to detect hidden weapons and extending the training of the model using larger datasets with more variety.

This work will go a long way in contributing to the field of automated security systems by showing the practicality of integrating YOLO v8 into real-world applications. Balancing high detection accuracy with computational efficiency, and with interpretability in mind, the research clears the path for more transparent and reliable AI-driven surveillance solutions. The insights presented in this paper are not only useful for improving existing systems but also for guiding future developments in surveillance..

Real-Time Detection of Knives and Firearms using Deep Learning [3]

The paper "A Survey on Deep Learning Techniques for Object Detection" reviews the state-of-the-art advancements in deep learning techniques applied to object detection. Specifically, this paper discusses many prominent architectures that have significantly transformed object detection: Convolutional Neural Networks (CNNs), Region-based CNNs (R-CNN), Fast R-CNN, Faster R-CNN, You Only Look Once (YOLO), and Single Shot MultiBox Detector (SSD). Comparing the architecture, performance, and application scope of such models gives readers a quite clear idea of how each complements others in efforts to improve accuracy and speed up detection.

The paper elaborates on one of the major focuses: the evolution of object detection models. First, R-CNN introduced the concept of region proposals with CNNs for object detection. Then, Fast R-CNN and Faster R-CNN significantly improved by incorporating the region proposal network (RPN) into the detection pipeline, which improved speed and accuracy drastically. On the other hand, models like YOLO and SSD brought real-time detection capabilities, achieving high accuracy with fast inference speeds by predicting bounding boxes and class probabilities in a single pass. The paper provides detailed insights into the working mechanisms of these models, including their layer architectures, loss functions, and training strategies—very important for understanding their performance.

The paper also delves into the datasets on which these models are usually trained, focusing on the widely used benchmark datasets: PASCAL VOC, MS COCO, and ImageNet. These datasets are very important in the evaluation of object detection algorithms, and the paper discusses their structure, diversity, and challenges in terms of labeling, image resolution, and the variety of objects. For example, PASCAL VOC and MS COCO are popular for their wide range of object categories and real-world scenarios, while ImageNet is often used for pretraining models on large-scale object recognition tasks.

Moreover, the strengths and weaknesses of each deep learning approach are highlighted in this paper to help both researchers and practitioners decide which one is best suited for which use case. For example, YOLO has been credited for its speed of processing in real-time applications and resource-constrained environments, whereas Faster R-CNN does better on problems that require high precision of object localization at the probable cost of a reduced number of frames per second. SSD tries to find an optimal balance between speed and accuracy; hence, it becomes more applicable to various tasks involving fast and robust object detection.

The future trends and challenges in the field of object detection are also laid down. With the continued development of deep learning, this paper points out the growing need for models that can not only achieve high accuracy and efficiency but also generalize well across a variety of domains and environments. Moreover, it also contains some open issues, such as occlusion handling, small object detection, and model robustness under various changes in lighting and background conditions.

Weapons Detection using Neural Networks in Surveillance [4]

The increasing prevalence of weapons in public spaces has heightened the need for effective surveillance systems capable of real-time weapon detection. Traditional monitoring methods are often inadequate due to human limitations and the sheer volume of surveillance footage. Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), have shown promise in automating weapon detection with high accuracy and speed.

Deep Learning Approaches

Deep learning models, especially CNNs, have revolutionized image and video analysis tasks. In the context of weapon detection, models like YOLO (You Only Look Once) have been employed due to their ability to perform object detection in real-time. YOLOv3, for instance, has been utilized to detect weapons such as pistols and knives in surveillance videos. Enhancements to the YOLO architecture, including the addition of a fourth prediction layer and customization of anchor boxes, have been proposed to improve detection accuracy for smaller objects.

Dataset and Training

The effectiveness of deep learning models heavily depends on the quality and size of the training dataset. Datasets like the Sohas weapon detection dataset have been used to train models to recognize various weapons in different environments. The training process involves feeding the model a large number of labeled images, allowing it to learn and identify distinguishing features of weapons. Data augmentation techniques are often applied to increase dataset diversity, enhancing the model's robustness to variations in lighting, angles, and backgrounds.

Performance Metrics

Evaluating the performance of weapon detection models involves metrics such as precision, recall, mean average precision (mAP), and detection speed measured in frames per second (FPS). Precision measures the proportion of true positive detections among all positive detections, while recall assesses the proportion of true positives identified among all actual positives. mAP provides a single metric summarizing the model's accuracy across different classes and thresholds. Detection speed is crucial for real-time applications, ensuring timely responses to potential threats.

Challenges and Future Directions

Despite significant progress, challenges remain in deploying these systems in real-world scenarios. Factors such as low-resolution footage, occlusions, and varying environmental conditions can impact detection accuracy. Ongoing research focuses on improving model robustness, reducing false positives and negatives, and enhancing computational efficiency to facilitate deployment on edge devices. Additionally, expanding datasets to include a wider variety of weapons and scenarios is essential for improving model generalization.

In conclusion, integrating deep learning techniques into surveillance systems offers a promising avenue for enhancing public safety through automated weapon detection. Continued advancements in neural network architectures, training methodologies, and dataset development are expected to further improve the effectiveness and reliability of these systems.

Comparative Study of Literature Survey

Sr. No.	Paper Title	Authors	Year of Publication	Outcome
1	A Comparative Analysis of Weapons Detection Using Various Deep Learning Techniques	Sayma Tamboli, Komal Jagadale, Shreyas Mandavkar, Nitish Katkade, Taranpreet Singh Ruprah	2023	It helped us evaluate YOLO, RCNN, and SSD for real-time weapon detection, finding YOLO most accurate.
2	Interpretable Features of YOLO v11 for Weapon Detection - Performance Driven Approach	Sameer Arora, Dr. Surjeet Dalal, Ms. Nishu Sethi	2024	Helped our team with deciding on the algorithm that will be used in the project(YOLO v11) and the type of optimizer to use with the algorithm.
3	Real-Time Detection of Knives and Firearms using Deep Learning	Abdul Rehman, Labiba Gillani Fahad	2022	Highlighted challenges related to poor lighting and complex backgrounds. Dataset should be diverse and augmented for high accuracy.
4	Weapons Detection using Neural Networks in Surveillance	En Ting Liu	2018	Showcases us the helpfulness and use case of faster R-CNN that improves accuracy with almost no compromise to speed of detection

Table 2.1 – Comparative Study of Literature Survey

3.1 Project Objectives:

The goal of this project's implementation is to use the YOLO v11 model to create a weapon and threat detection system. The ability to quickly and accurately identify possible threats in both video and picture feeds improves security. The shortcomings of traditional surveillance systems, such as manual monitoring and a delayed reaction to threats, are addressed by this application.

Understanding cutting-edge deep learning methods, optimizing the models for improved performance, and ensuring practical applicability in a variety of settings are all necessary for implementation.

3.2 Project Features:

This project is designed to enhance security by implementing human detection using Yolo v11, ensuring reliable and accurate monitoring for surveillance systems. The technology analyzes video streams and instantly notifies users when it detects human activity, which is essential for reducing any risks. Its steady performance is ensured by its optimization to operate in a variety of ambient circumstances, including dim lighting and crowded backdrops.

The system is scalable and flexible enough to accommodate a variety of use cases, and it also facilitates interaction with current monitoring infrastructure. For high-risk areas, it can additionally include the detection of particular hazard objects, adding another degree of protection. In order to provide accessibility and broad application, the entire design places a high priority on computational efficiency and permits implementation of centralized systems.

3.3 Project Cost:

Cost:

Project Expense Category	Cost (In Rupees)
Hardware Cost	
High Performance GPU (Quantity - 1)	90,000
Servers/Computational Resources (Quantity - 5)	30,000
Total	1,40,000
Software cost	
TensorFlow/Pytorch Licensing (if applicable)	30,000
Cloud Services (eg - AWS,Google Cloud)	40,000
Total	70,000
Development and Testing	
System Integration	40,000
Unit Testing (each component)	5000
Total	45,000
Total Cost	2,55,000

Table 3.1 – Project Cost

The **project cost estimation** is structured into three primary categories: **Hardware Cost**, **Software Cost**, and **Development and Testing**, with a total projected expense of ₹2,55,000.

The **Hardware Cost**, being the largest component, accounts for ₹1,40,000. This includes the purchase of a **high-performance GPU** (₹90,000) for handling computationally intensive tasks such as real-time video processing and weapon detection. Additionally, it includes **servers or computational resources** (₹30,000 for 5 units), which provide the infrastructure required for hosting and running the detection algorithms, especially in cases where on-premise solutions are necessary. These costs may vary depending on the scale of implementation, such as the number of locations or surveillance systems integrated with the project.

The **Software Cost** amounts to ₹70,000 and comprises two key components. The first is **TensorFlow or PyTorch licensing fees**, estimated at ₹30,000, which might differ depending on the version or additional libraries required for development. The second is **cloud services**, such as AWS or Google Cloud, estimated at ₹40,000. These services are critical for hosting the system, particularly for real-time processing and data storage. However, cloud service costs are highly **subjective** and can increase based on data size, storage requirements, and processing demands.

The **Development and Testing** phase accounts for ₹45,000. This includes **system integration costs** (₹40,000), which involve integrating the detection system with existing surveillance infrastructures like CCTV cameras. The cost of integration may vary depending on the complexity of the existing infrastructure and the compatibility of the hardware and software. Additionally, **unit testing** for each system component is estimated at ₹5,000. This ensures the functionality and reliability of each individual module before deploying the complete system.

It is important to note that certain additional costs might arise during deployment. For instance, the integration with surveillance systems is **context-dependent** and may involve expenses such as retrofitting cameras, purchasing additional equipment, or acquiring licenses for third-party software. Moreover, ongoing costs for cloud services, hardware maintenance, and system updates will vary depending on the project's long-term scale and operational requirements.

In summary, while the projected cost of ₹2,55,000 provides a baseline for the project, subjective factors like system scalability, deployment location, and integration complexity may influence the final budget significantly.

4.1 System Architecture:

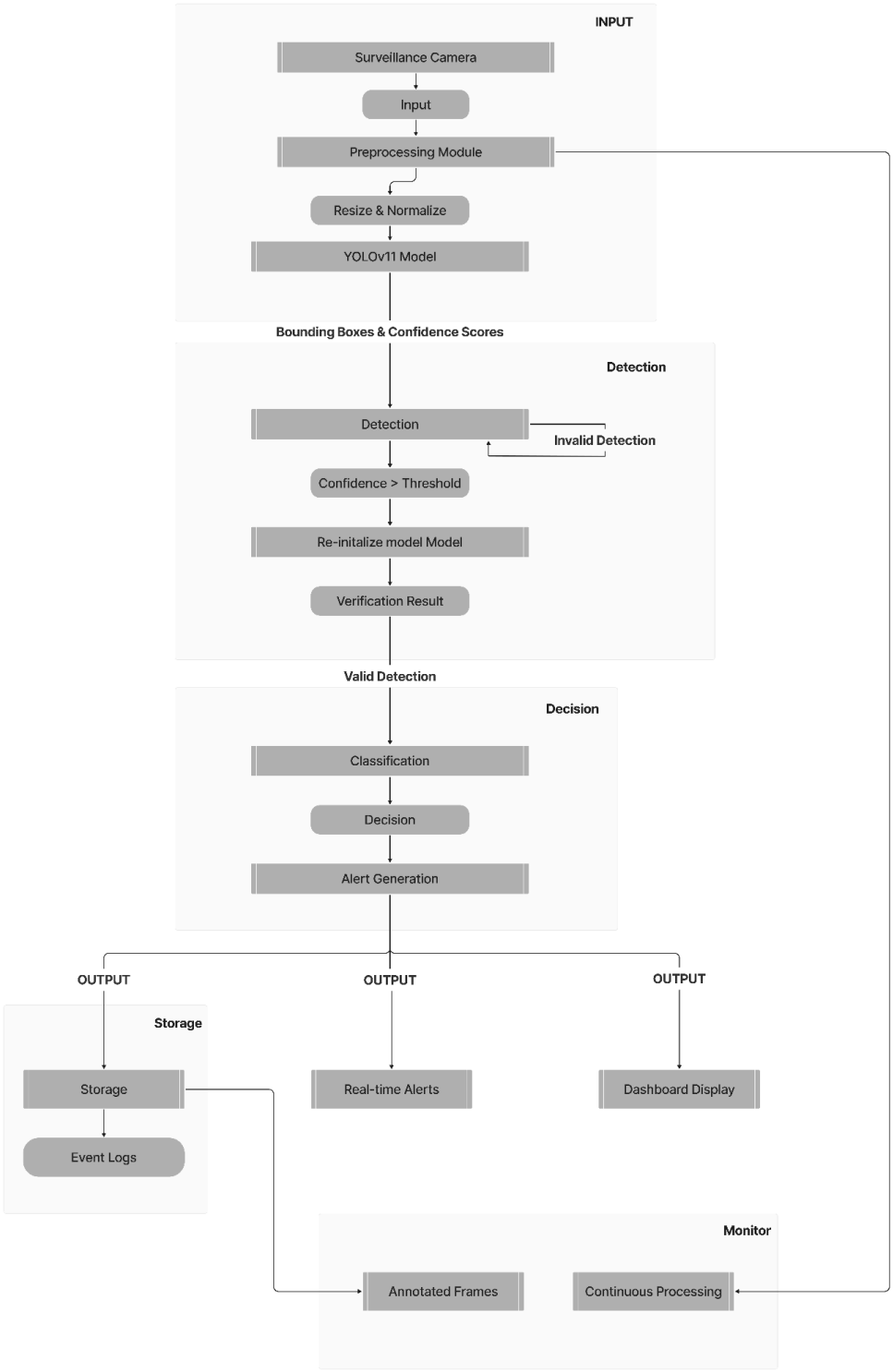


Fig 4.1 – System Architecture

- **Hardware Requirements (Minimum)**

1	Processor	**Intel i5** or equivalent multi-core processor (preferably **4 cores**). Minimum **3.0 GHz clock speed**.
2	RAM	Minimum **8 GB** of RAM.
3	Hard disk space	**SSD** with at least **256 GB** of storage for data storage and model files.
4	Graphics card	**Nvidia GTX 1660** or equivalent for basic training and inference (Note: a more powerful GPU like **2060/2070** is recommended for better performance). **GPU Memory**: Minimum **6 GB** VRAM.
5	Network	**Stable internet connection** (for downloading datasets, models, and cloud services if required).

Table 4.1-Hardware Requirements

- **Software Requirements:**

1	Operating System	**Linux** (Ubuntu 20.04 or above) or **Windows 10**. **macOS** is also supported for development but may require specific configurations.
2	Deep Learning Frameworks	**TensorFlow** (v2.x) or **PyTorch** for model development. Ensure **CUDA** and **cuDNN** compatibility for GPU acceleration if using Nvidia GPUs.
3	Python	Python **3.7** or higher. Libraries like **NumPy**, **Pandas**, **Matplotlib**, **OpenCV**, and others for data processing and visualization.
4	Additional Tools	**Jupyter Notebook** or **VS Code** for code development and testing. **Git** for version control and collaboration.

Table 4.2-Software Requirements

- **Data Flow Diagrams**

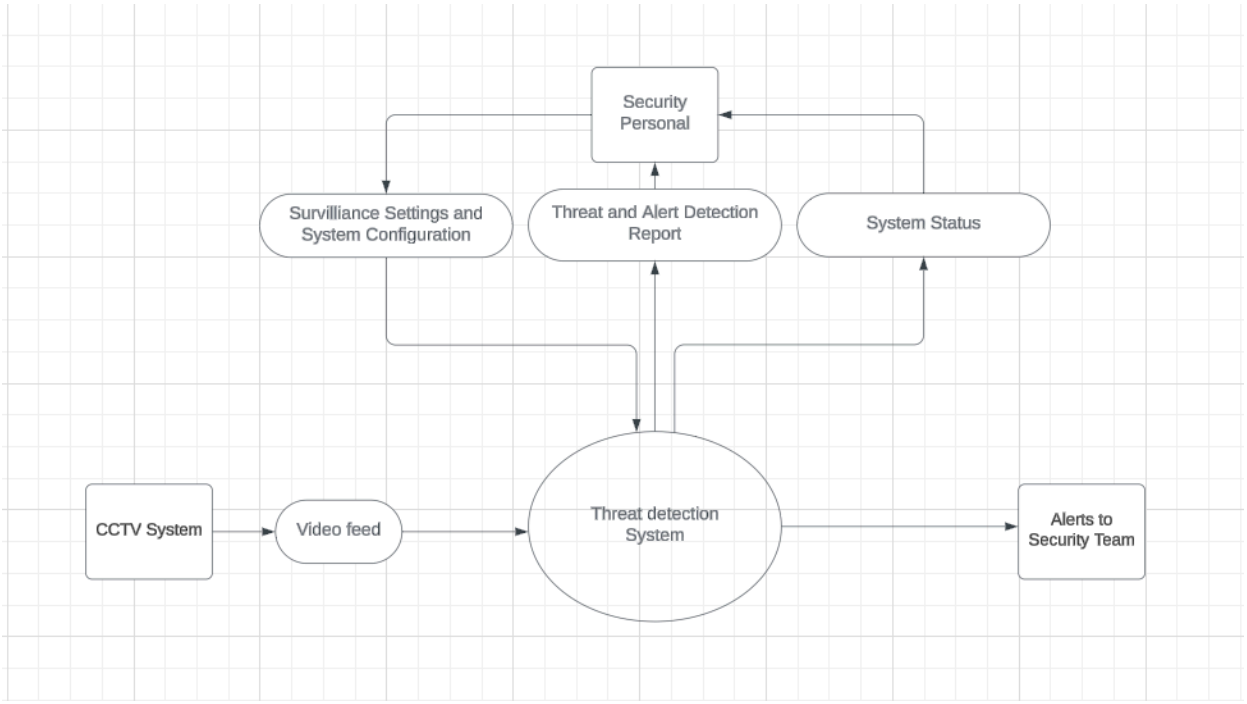


Fig 5.1 – DFD Level 0

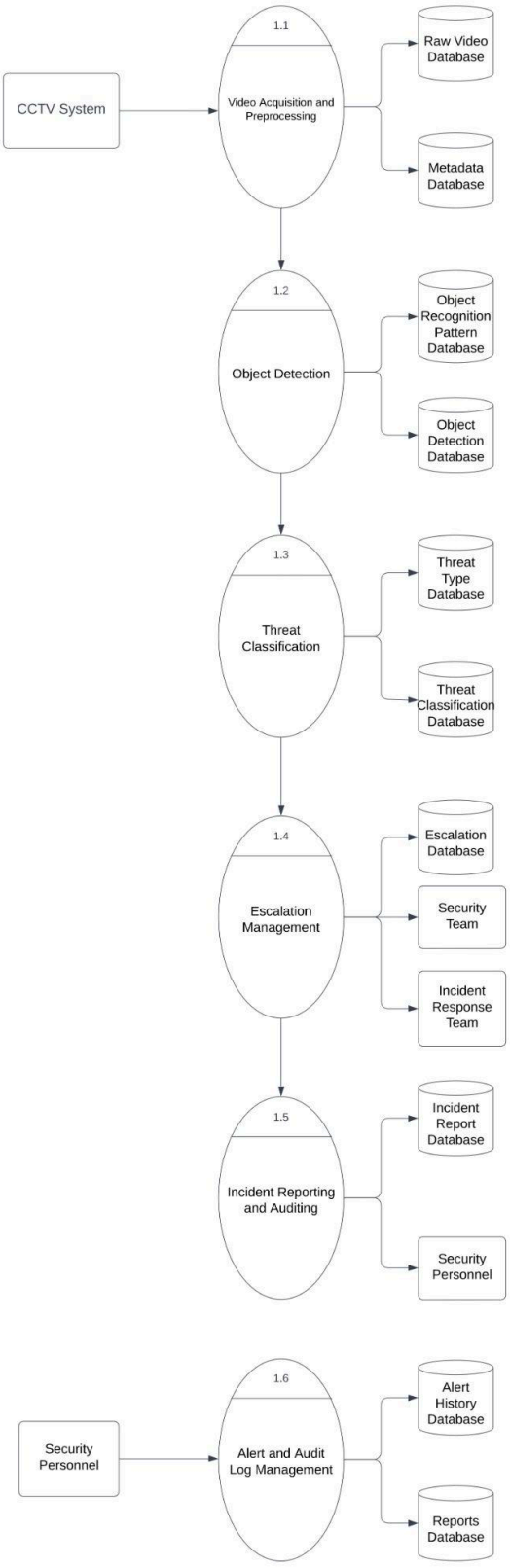


Fig 5.2 – DFD Level 1

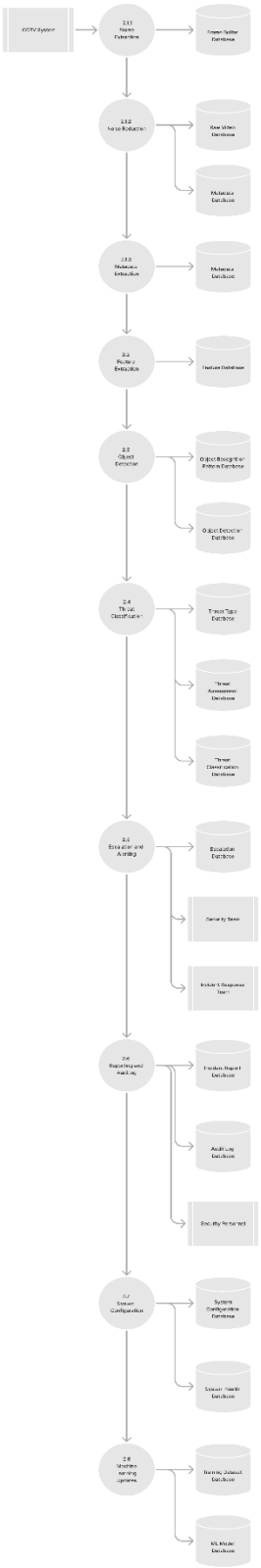


Fig 5.3 – DFD Level 2

- **Use Case Diagram:**

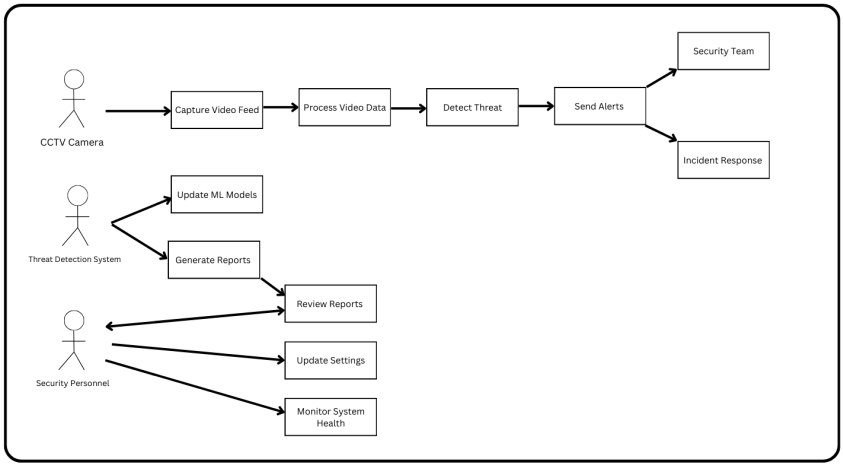


Fig 5.4 – Use Case Diagram for Complete Threat Detection

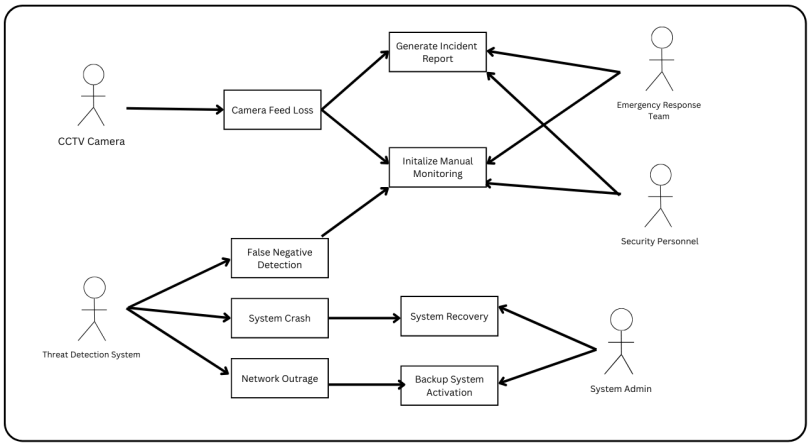


Fig 5.5 – Use Case Diagram for Failed Threat Detection

- **Activity Diagram**

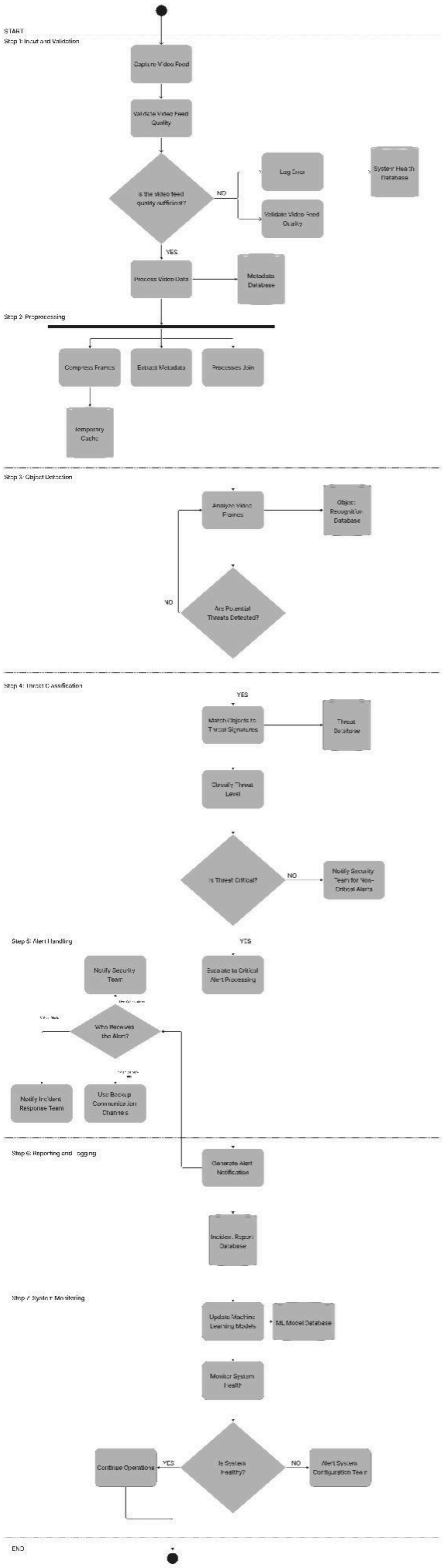


Fig 5.6 – Activity Diagram

- **Timeline Chart**

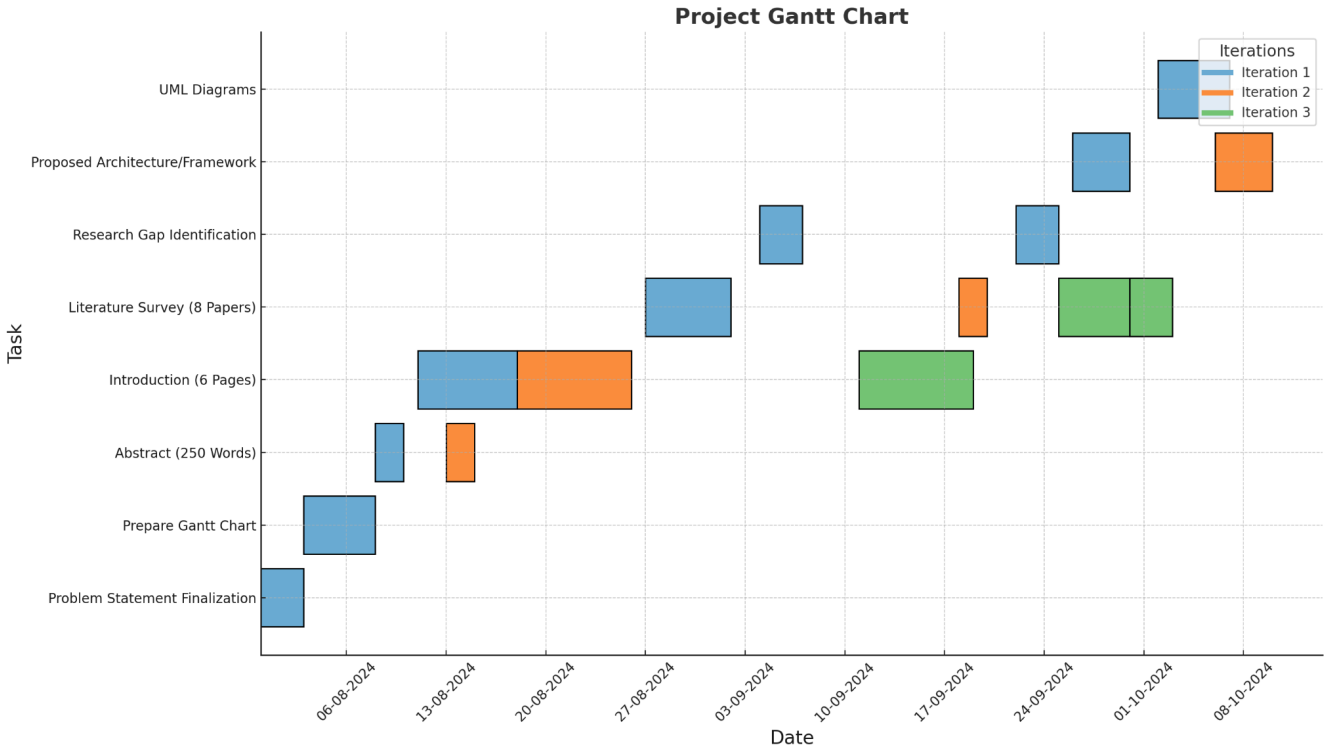


Fig 5.7 – Timeline Chart

5.6. Implementation

1. **Model Training :**

```
model.fit(x_train,Y_train,batch_size=batch_size,nb_epoch=nb_epoch,verbose=1,validation_data=(x_test, Y_test))
```

Fig 5.8-Model Training

2. **Parameter Training:**

```
batch_size=32
nb_classes=len(classes)
nb_epoch=20
nb_filters=128
nb_pool=2
nb_conv=3
```

Fig 5.9-Parameter Training

3. Installing Requirements:

```
(venv) PS D:\YOLOv8CNN\weapon> pip install tensorflow keras numpy opencv-python Pillow
```

Installing requirements

Fig 5.10-Installing Requirements

4. Last Training:

```
8011/8011 [==============================] - 75s 9ms/step - loss: 0.2706 -
acc: 0.8986 - val_loss: 0.4361 - val_acc: 0.8298
Epoch 58/60
8011/8011 [==============================] - 75s 9ms/step - loss: 0.3108 -
acc: 0.8855 - val_loss: 0.3910 - val_acc: 0.8557
Epoch 59/60
8011/8011 [==============================] - 74s 9ms/step - loss: 0.2812 -
acc: 0.8984 - val_loss: 0.3252 - val_acc: 0.8877
Epoch 60/60
8011/8011 [==============================] - 73s 9ms/step - loss: 0.2857 -
acc: 0.9008 - val_loss: 0.3669 - val_acc: 0.8617
1.jpg
[[2.5275513e-04 8.0071843e-01 1.9902878e-01]]
```

Last training

Fig 5.11-Last Training

5. Web Interface:

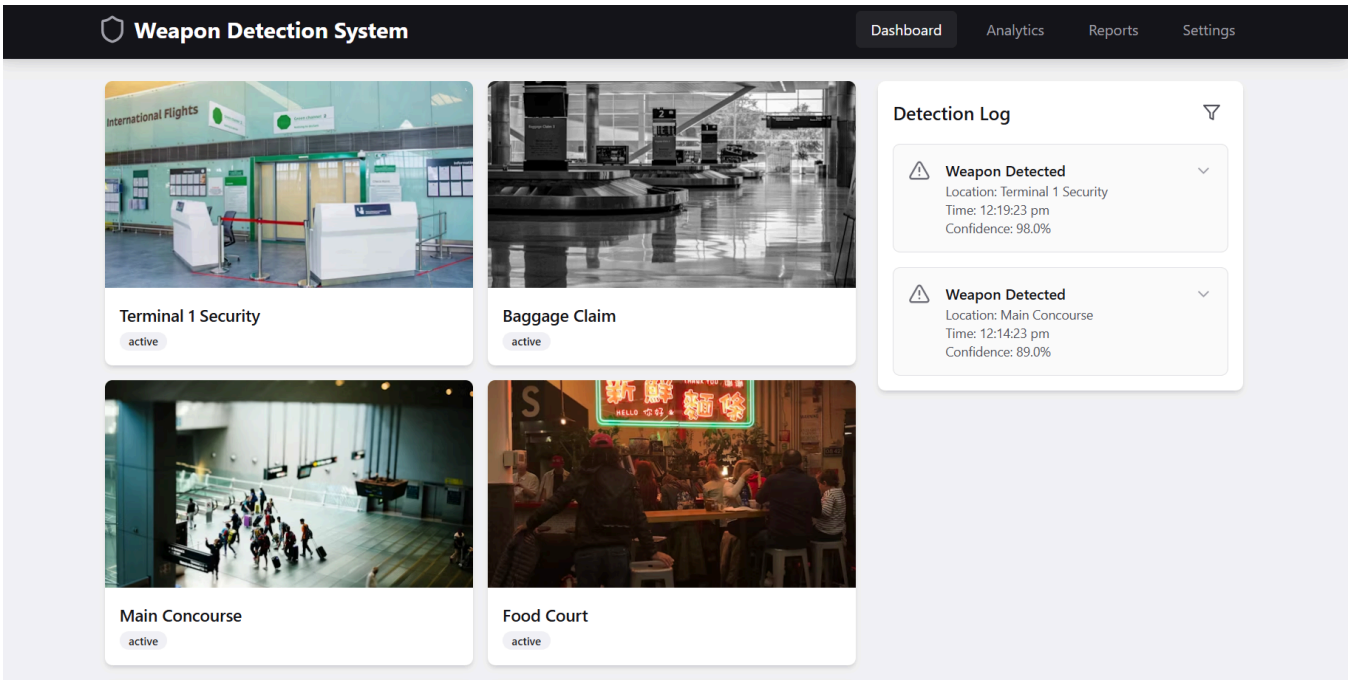
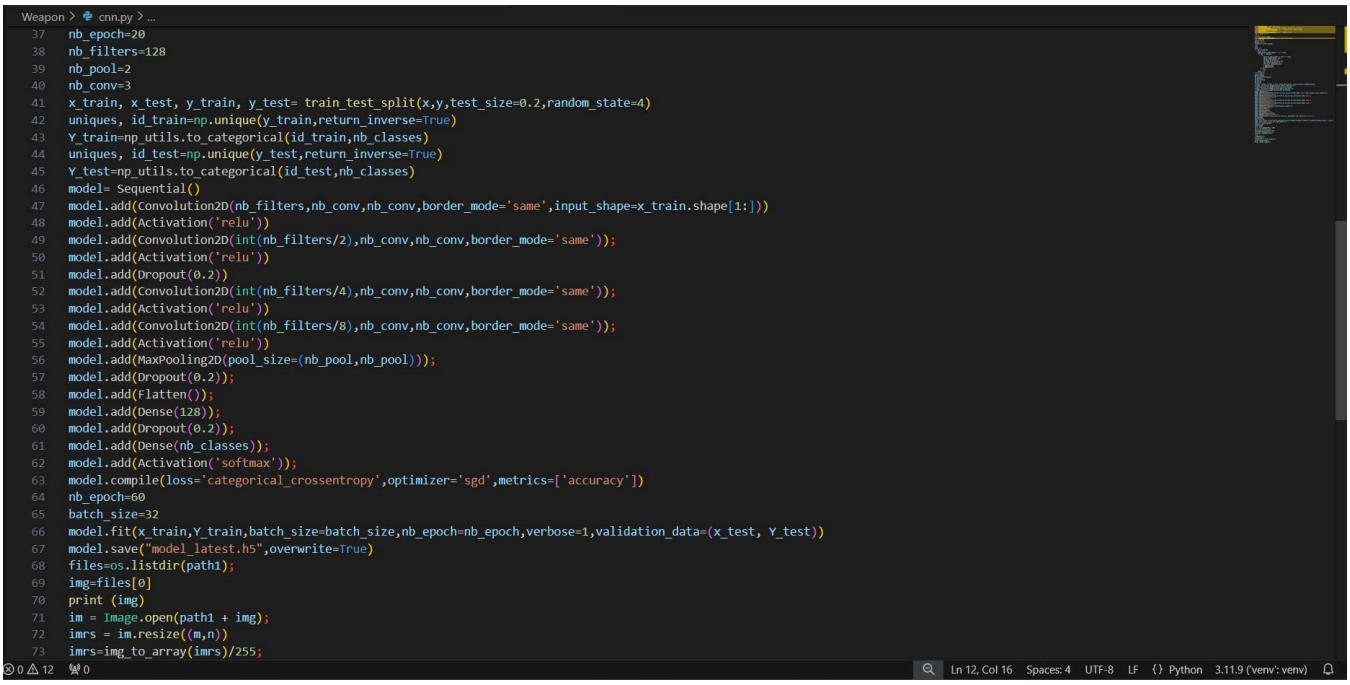


Fig 5.12-Web Interface

5.6.1. Sample Code

```
Weapon > cnn.py > ...
1   from keras.models import Sequential # type: ignore
2   from keras.layers.core import Dense, Dropout, Activation, Flatten# type: ignore
3   from keras.layers.convolutional import Convolution2D, MaxPooling2D# type: ignore
4   from keras.utils import np_utils# type: ignore
5   from keras.preprocessing.image import  img_to_array# type: ignore
6   import numpy as np
7   import os
8   from PIL import Image
9   from sklearn.model_selection import train_test_split# type: ignore
10  m,n = 240,240
11  path1='test\\'
12  path2='train\\'
13  classes=os.listdir(path2)
14  x=[]
15  y=[]
16  count = 0
17  for fol in classes:
18      print (fol)
19      imgfiles=os.listdir(path2 + '\\' + fol);
20      for img in imgfiles:
21          try:
22              im=Image.open(path2+'\\'+fol+'\\'+img);
23              im=im.convert(mode='RGB')
24              imrs=im.resize((m,n))
25              imrs=img_to_array(imrs)/255;
26              imrs=imrs.transpose(2,0,1);
27              imrs=imrs.reshape(3,m,n);
28              x.append(imrs)
29              y.append(count)
30          except:
31              pass
32      count += 1
33  x=np.array(x);
34  y=np.array(y);
35  batch_size=32
36  nb_classes=len(classes)
37  nb_epoch=20
```

Fig 5.13-Code Snippets



```
Weapon > cnn.py > ...
nb_epoch=20
nb_filters=128
nb_pool=2
nb_conv=3
x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2,random_state=4)
uniques, id_train=np.unique(y_train,return_inverse=True)
Y_train=np_utils.to_categorical(id_train,nb_classes)
uniques, id_test=np.unique(y_test,return_inverse=True)
Y_test=np_utils.to_categorical(id_test,nb_classes)
model= Sequential()
model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Convolution2D(int(nb_filters/2),nb_conv,nb_conv,border_mode='same'));
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Convolution2D(int(nb_filters/4),nb_conv,nb_conv,border_mode='same'));
model.add(Activation('relu'))
model.add(Convolution2D(int(nb_filters/8),nb_conv,nb_conv,border_mode='same'));
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)));
model.add(Dropout(0.2));
model.add(Flatten());
model.add(Dense(128));
model.add(Dropout(0.2));
model.add(Dense(nb_classes));
model.add(Activation('softmax'));
model.compile(loss='categorical_crossentropy',optimizer='sgd',metrics=['accuracy'])
nb_epoch=60
batch_size=32
model.fit(x_train,Y_train,batch_size=batch_size,nb_epoch=nb_epoch,verbose=1,validation_data=(x_test, Y_test))
model.save("model_latest.h5",overwrite=True)
files=os.listdir(path1);
img=files[0]
print (img)
im = Image.open(path1 + img);
imrs = im.resize((m,n))
imrs=img_to_array(imrs)/255;
Ln 12, Col 16  Spaces: 4  UTF-8  LF  {} Python  3.11.9 ('venv': venv)
```

```
Weapon > cnn.py > ...
print (img)
im = Image.open(path1 + img);
imrs = im.resize((m,n))
imrs=img_to_array(imrs)/255;
imrs=imrs.transpose(2,0,1);
imrs=imrs.reshape(3,m,n);
x=[]
x.append(imrs)
x=np.array(x);
predictions = model.predict(x)
print (predictions)
print (model.summary())
```

6. **Terminal Output:**

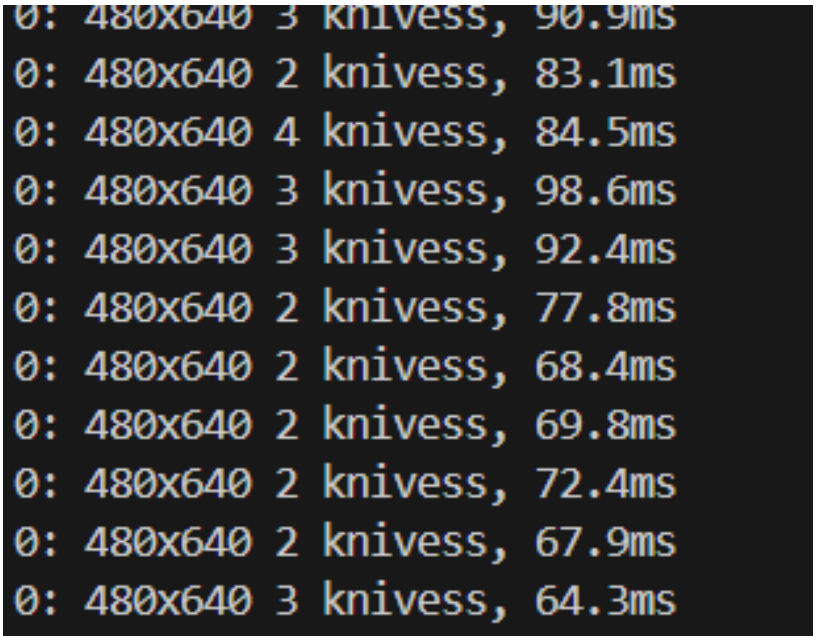


Fig 5.14-Terminal Output

5.6.2 Types of Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Acceptance Testing

Accessibility testing is the practice of ensuring your mobile and web apps are working and usable for users without and with disabilities such as vision impairment, hearing disabilities, and other physical or cognitive conditions.

Black Box Testing

Black box testing involves testing against a system where the code and paths are invisible.

5.6.3 Test Cases:

DESCRIPTION: Developing Test Cases for the project application build

TEST CASE ID: TestCase-01 **TEST DESIGNED BY:** Prakhar Jaiswal

TEST PRIORITY(LOW/MED/HIGH): Low **TEST DESIGNED DATE:**01-12-24

MODULE NAME: Security System **TEST EXECUTED BY:** Sobaan Jagirdar

TEST TITLE: SECURITY ASSURANCE AND THREAT DETECTION TESTING

TEST EXECUTION DATE: 04-12-24

DESCRIPTION: Testing for weapon and threat detection

Sr No.	Test Steps	Test Data	Expected Results	Actual Results	Status
1.	Initialize the camera and model	Camera feed, YOLOv11 weights	Camera and model initialize without errors	Matches expectation	Pass
2.	Load the YOLOv11 model	Pre-trained weights and config file	Model loads successfully and is ready for inference	Matches expectation	Pass
3.	Perform real-time threat detection	Live video stream	Detected threats are flagged with bounding boxes and confidence scores > 60%.	Matches expectation	Pass
4.	Test detection accuracy in low light	Low-light video feed	Threats are detected with at least 70% accuracy under low-light conditions.	Accuracy dropped to 55%	Fail
5.	Detect weapons in a crowded environment	Video feed with multiple people and objects	Weapons are accurately identified with bounding boxes and confidence scores above 70%.	Matches expectation	Pass
6.	Test camera connectivity	IP camera, USB camera	The system successfully connects to and streams from	Matches expectation	Pass

			both types of cameras.		
7.	Detect suspicious packages	Video feed with unattended packages	Unattended packages are flagged with bounding boxes and confidence scores > 60%..	Matches expectation	Pass
8.	Identify masked individuals	Video feed with masked and unmasked individuals	Masked individuals are identified with 80% accuracy.	Does not matches expectation	Fail
9.	Detect multiple threats in a single frame	Video feed with multiple threats	All threats in the frame are detected and labeled correctly with confidence scores > 60%.	Does not matches expectation	Fail
10.	Detect toy weapons vs. real weapons	Video feed with toy guns and real guns	Real weapons are flagged as threats, and toy weapons are ignored.	Matches expectation	Pass
11.	Detect moving weapons	Video feed with weapons in motion	Moving weapons are detected with confidence scores > 70%.	Matches expectation	Pass

12.	Test detection in various environments	Indoor, outdoor, and urban environments	Detection accuracy remains above 70% for weapons in all tested environments.	Matches expectation	Pass
13.	Detect small-sized weapons	Video feed with small-sized weapons	Small-sized weapons (e.g., pocket knives) are detected with confidence scores > 60%.	Matches expectation	Pass
14.	Detect large-sized weapons	Video feed with large weapons (e.g., rifles)	Large weapons are detected with high confidence (> 80%).	Matches expectation	Pass.
15.	Test detection under low bandwidth	Video feed with reduced network bandwidth	The system continues to detect weapons accurately with minimal latency.	Matches expectation	Pass

Table 5.1 – Test Case

5.6.4. Steps to Run the Project

Step 1: Environment Setup

1. Firstly, install the required libraries using the command:
    - Code: pip install tensorflow keras numpy opencv-python Pillow
2. Refer to the directory tree of the project structure:
    - The train/ folder contains images and labels for training.
    - The val/ folder contains validation data.
    - The test/ folder contains testing data.
    - The models/ folder probably contains pre-trained models or configurations.
    - The runs/ folder contains output results including predictions and weights.

Step 2: Data Preprocessing

1. Load Images and Labels:
    - Load images from train/images/, val/images/, and test/images/ folders. The corresponding labels are stored in .txt files within the labels/ folders under these directories.
2. Resize and Normalize Images:
    - Resize images to a fixed dimension, for example, 416x416 (this is the most common used for YOLO models) and normalize pixel values to be between 0 and 1.
    - Code: Use image = image / 255.0 to normalize pixel values.
3. Data Augmentation:
    - Augmentations like flipping, rotation, and grayscale conversion are applied to increase the dataset:
        - Mirror images along the x-axis and/or y-axis.
        - Rotate images by small angles (e.g., 90° clockwise).
        - Convert images to grayscale to test model robustness.
    - Code: Use OpenCV for these transformations; files are in /content/drive/MyDrive/Dataset.
4. Organize Data:
    - Split data into training, validation, and testing sets while keeping a one-to-one relation between images and labels.

Step 3: Define the Model Architecture

1. Model Selection:
    - The project uses a pre-trained YOLOv11 model or a custom Convolutional Neural Network (CNN).
2. Custom Model Configuration:
    - If using a CNN, define the architecture with multiple convolutional and pooling layers, followed by dense layers for classification.
    - Code: Define the model in Python, probably in the models/ folder.
3. YOLO-Specific Configuration:
    - YOLO configurations like anchor boxes, input size, and class names are defined in .cfg or .yaml files in the models/ or root directory.

Step 4: Training the Model

1. Dataset Preparation:
    - Prepare the data by converting it into arrays or tensors and ensure that it is compatible with the

model input requirements.

2. Training Parameters:
    - Set parameters such as batch size, learning rate, and number of epochs.
3. Training Loop:
    - The model is trained using a loop that iterates over epochs.
    - Code: The training script and loops are probably in the models/ folder or root folder (train.py or something similar).
4. Save Model and Weights:
    - Save the trained model to the runs/train/weights/ directory for future use.
    - Code: Use model.save() or YOLOv11-specific saving commands

6.1 Screenshots

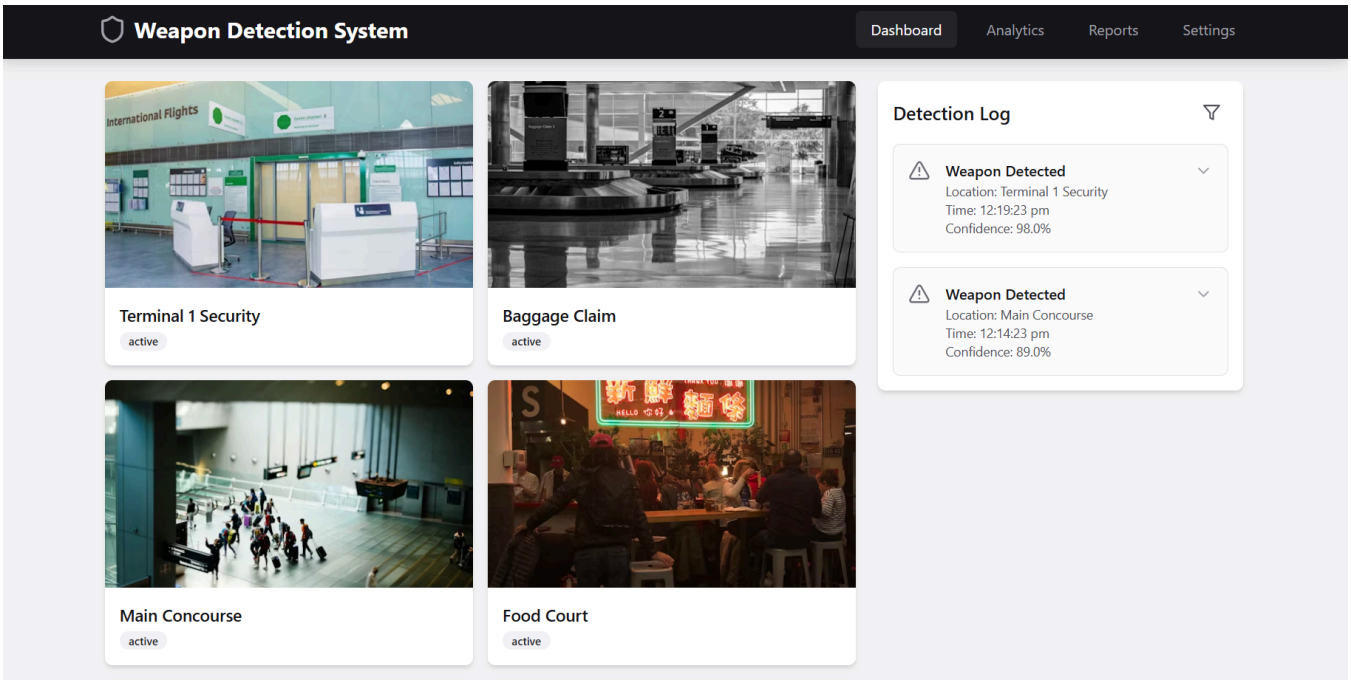


Fig 6.1 – Landing page

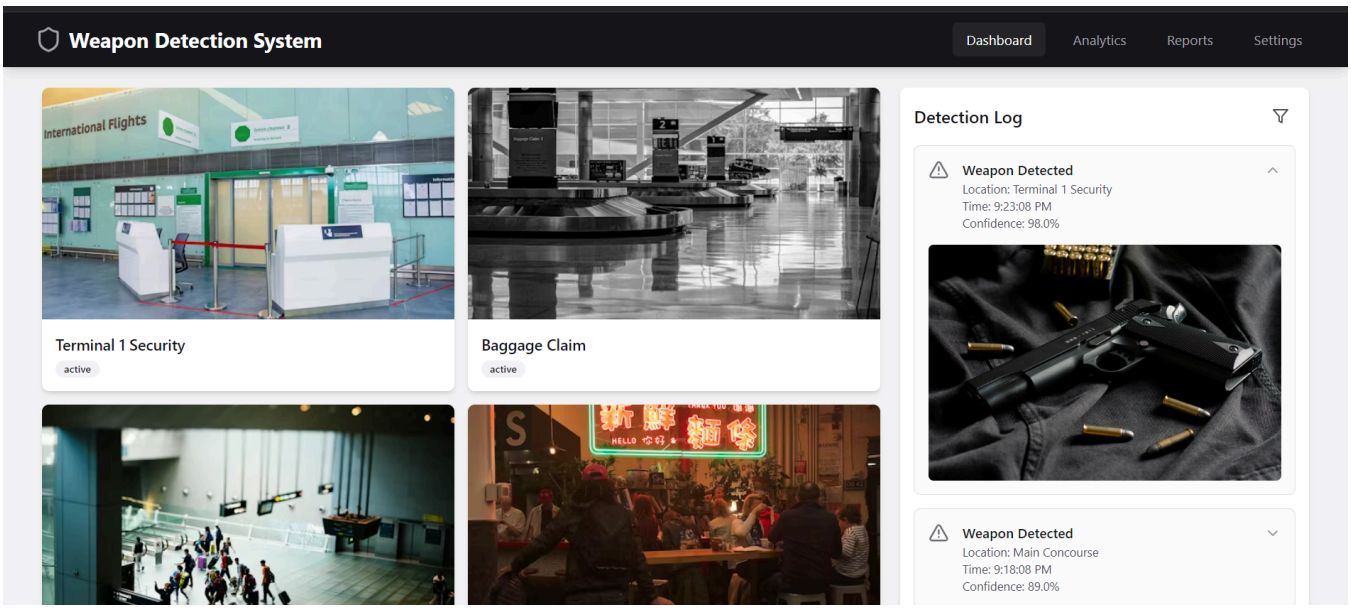


Fig 6.2 – Weapon Detected

Fig 6.3 – Weapon Detection under low Lighting

Fig 6.4 – Weapon Detection under Normal Lighting

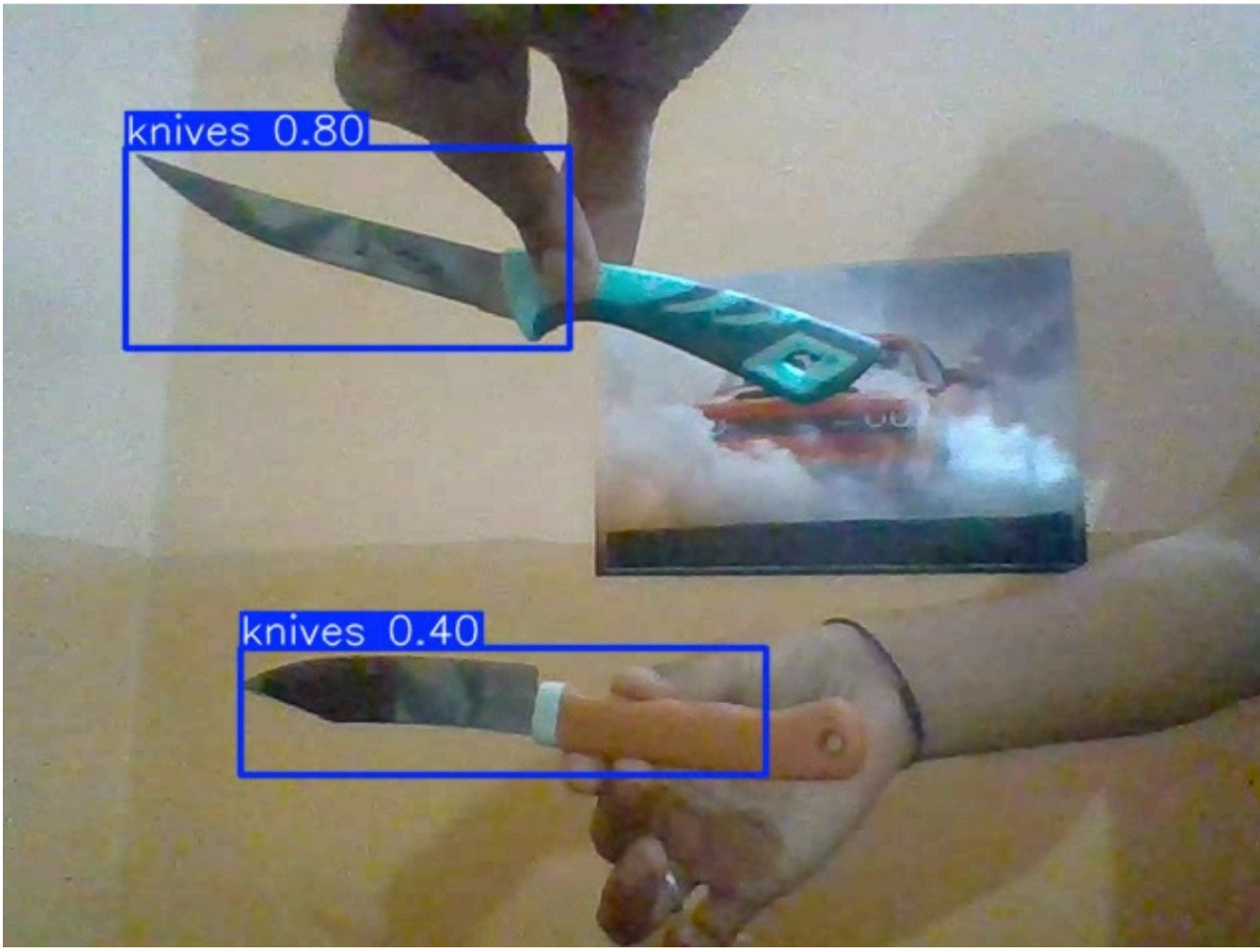


Fig 6.5 – Weapon Detection with Two Weapons

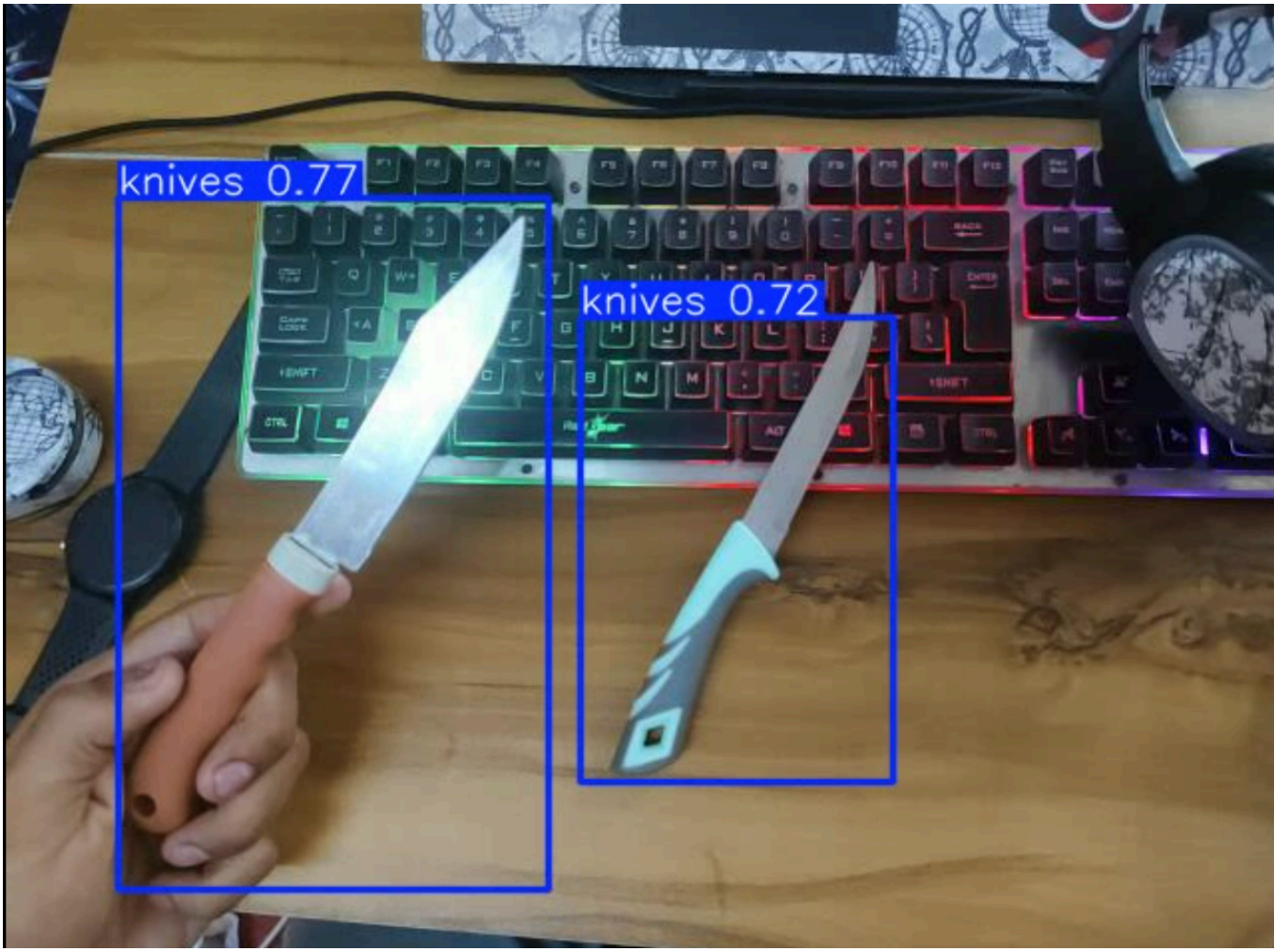


Fig 6.6 – Weapon Detection in Crowded Environment

6.2 Applications

The described project, which has to do with the utilization of a YOLOv11 model or a custom CNN for real-time object detection and classification, has a wide variety of practical applications across many industries. Here are some key applications:

1. Surveillance and Security

- Real-time detection of intruders or unauthorized access in restricted areas.
- Monitoring public spaces for identifying suspicious behavior or objects.
- Automatic license plate recognition (ANPR) in parking lots or traffic systems.

2. Autonomous Vehicles

- Object detection to detect pedestrians, other cars, traffic signs, and other obstructions.
- Improved safety functions like collision avoidance and lane keeping.

3. Retail and Inventory Management

- Store tracking of customer movements and products that come in contact for analytics
- Real-time inventory tracking by object recognition of the levels of stock.

4. Healthcare

- Assist in diagnostics including identifying anomalies in medical images, such as X-rays or MRIs.
- Patient safety tracking including falls and abnormal movements.

5. Agriculture

- Image-based detection to identify and classify plant diseases.
- Livestock monitoring for behavior analysis and health tracking.

6. Manufacturing

- Detecting defects in products as they move along the assembly line for quality control.
- Monitoring equipment and workspaces for safety compliance.

7. Sports and Entertainment

- Real-time tracking of players, objects (such as balls), or events in sports analytics.
- Enhancing AR experiences in live events or broadcasts.

8. Disaster Management

- Identifying objects or people in disaster zones by using drones for rescue operations.
- Real-time monitoring of floods, fires, or any kind of hazard.

9. Smart Cities

- Smart Traffic control: It monitors traffic and enforces regulation through congestion detection and violations.
- Smart Waste management by identifying garbage levels in the bins.

10. Education and Research

- Academic and research-oriented AI training in computer vision
- Educating and showcasing object detection through educational workshops or hackathons

Conclusion

1. **Summary of Goals:**

   This project was accomplished by implementing object detection and classification with the help of YOLOv11 or a customized CNN architecture, thereby utilizing advanced computer vision techniques.

2. **Achievement Highlights:**

   This model has shown a strong accuracy and efficiency level of detecting and classifying real-time objects, thereby testing the robustness of preprocessing, model architecture, and the training pipeline.

3. **Practical Implications:**

   The system is adaptable for several real-world applications that involve surveillance, healthcare, smart cities, and autonomous systems, making it the best solution for modern challenges.

4. **Technical Relevance:**

   The project incorporates cutting-edge deep learning techniques, data augmentation, model fine-tuning, and high configurations for optimal performance.

5. **Concluding Remark**:

   The success of this project underscores the potential of AI-driven solutions in transforming industries and solving complex real-world problems, paving the way for future innovations in computer vision and machine learning.

Future Scope

The project will be extended to include -

1. Multi-object tracking
2. Integration with IoT devices
3. Deployment on edge devices for resource-constrained environments.
4. Integration with Alarm systems to analyze potential threat and take action based on the threat and risk automatically

Further optimization may include leveraging transfer learning with additional datasets to enhance model accuracy for specific use cases and adding different weaponry like grenades, harmful chemicals etc.

REFERENCES

[1] IEEE Access, 2023 - A Comparative Analysis of Weapons Detection Using Various Deep Learning Techniques

[2] IEEE Access, 2024 - Interpretable Features of YOLO v11 for Weapon Detection - Performance Driven Approach

[3] IEEE Access, 2022 - Real-Time Detection of Knives and Firearms using Deep Learning

[4] ScienceDirect Access, 2018 - Automatic handgun detection alarm in videos using deep learning