11/9/2023

Arming Against Violence

YOLO-Based Weapon Detection

Team - 592720
Ashwin Parthasarathy, 21BCE1641
Kaustubha Nandakishore, 21BPS1290
Pranshu Tyagi, 21BCE1271

1. Introduction

1.1 Project Overview

The rapid advancement of technology has given rise to various challenges in ensuring safety and security, particularly in public spaces and critical infrastructure. The detection of weapons, a critical aspect of security, has become an increasingly important concern. To address this challenge, this project focuses on the development of an efficient weapon detection system using the state-of-the-art YOLOv8 (You Only Look Once version 8) model in machine learning.

Our weapon detection system utilizes YOLOv8's advanced object detection capabilities to identify and localize weapons within images and video streams. By leveraging deep learning techniques and the power of YOLOv8, we aim to create a robust and real-time solution for enhancing security in a wide range of scenarios, from public transportation to high-security facilities.

The weapon detection application has been made by integrating the YOLOv8 object detection model with OpenCV and deployed with the help of an open-source Python library called **Streamlit**.

1.2 Purpose

The primary purpose of this project is to develop an advanced weapon detection system using the YOLOv8 model to enhance public safety and security. The project aims to address the pressing need for effective and efficient solutions for identifying and alerting authorities to the presence of weapons in various environments. By harnessing the power of Deep Learning and Computer Vision, this project intends to:

1. Improve Public Safety: Create a system capable of swiftly and accurately detecting weapons in real-time, thus aiding law enforcement and security personnel in proactively responding to potential threats and preventing acts of violence.

2. Support Law Enforcement: Provide law enforcement agencies with a valuable tool to enhance their capabilities in identifying concealed weapons, both in public spaces and at critical checkpoints.

3. Secure Sensitive Locations: Safeguard sensitive areas such as airports, government buildings, schools, and public events by implementing a reliable weapon detection system.

4. Facilitate Proactive Monitoring: Support the proactive monitoring of video feeds from surveillance cameras to identify potential threats and intervene as necessary.

5. Contribute to Crime Prevention: By deterring individuals from carrying weapons and reducing their chances of going undetected, the project indirectly contributes to deterring criminal activity.

6. Promote Ethical Use of AI: Advocate for the responsible and ethical use of artificial intelligence in public security by addressing potential biases and privacy concerns.

2. Literature Survey

2.1 Existing Problem

Weapon detection has become a pressing concern in contemporary society, given the potential threats to public safety and security. As technology has evolved, so have the methods for addressing this challenge.

Traditional weapon detection methods primarily rely on manual inspection and security personnel to identify weapons in public spaces, such as airports, malls, and government buildings. These methods, while effective to some extent, suffer from limitations in terms of speed and scalability. Human error and fatigue can also impact the accuracy of weapon detection using traditional methods.

In recent years, Computer Vision and Deep Learning have emerged as powerful tools for object detection, including weapons. Several techniques based on Convolutional Neural Networks (CNNs) and object detection models have been explored. Among these, YOLO (You Only Look Once) models have gained popularity due to their real-time detection capabilities.

Prior work has used YOLO-based models to detect weapons in images and videos. YOLO offers the advantage of faster inference times and improved accuracy, making it a suitable choice for real-time applications. The YOLOv8 model, the focus of our project, represents the latest version of the YOLO series and is expected to provide enhanced performance.

2.2 References

- The YOLOv8 Github Repository: https://github.com/ultralytics/ultralytics/tree/main
- Smartinternz Google Colab Notebooks
- https://stackoverflow.com/questions/75983653/how-to-save-a-yolov8-model-after-some-training-on-a-custom-dataset-to-continue-t
- https://github.com/CodingMantras/yolov8-streamlit-detection-tracking/tree/master
- https://www.youtube.com/watch?v=WbomGeoOT_k
- https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/
- https://www.kaggle.com/datasets/snehilsanyal/weapon-detection-test?select=weapon_detection
- https://streamlit.io/

2.3 Problem Statement Definition

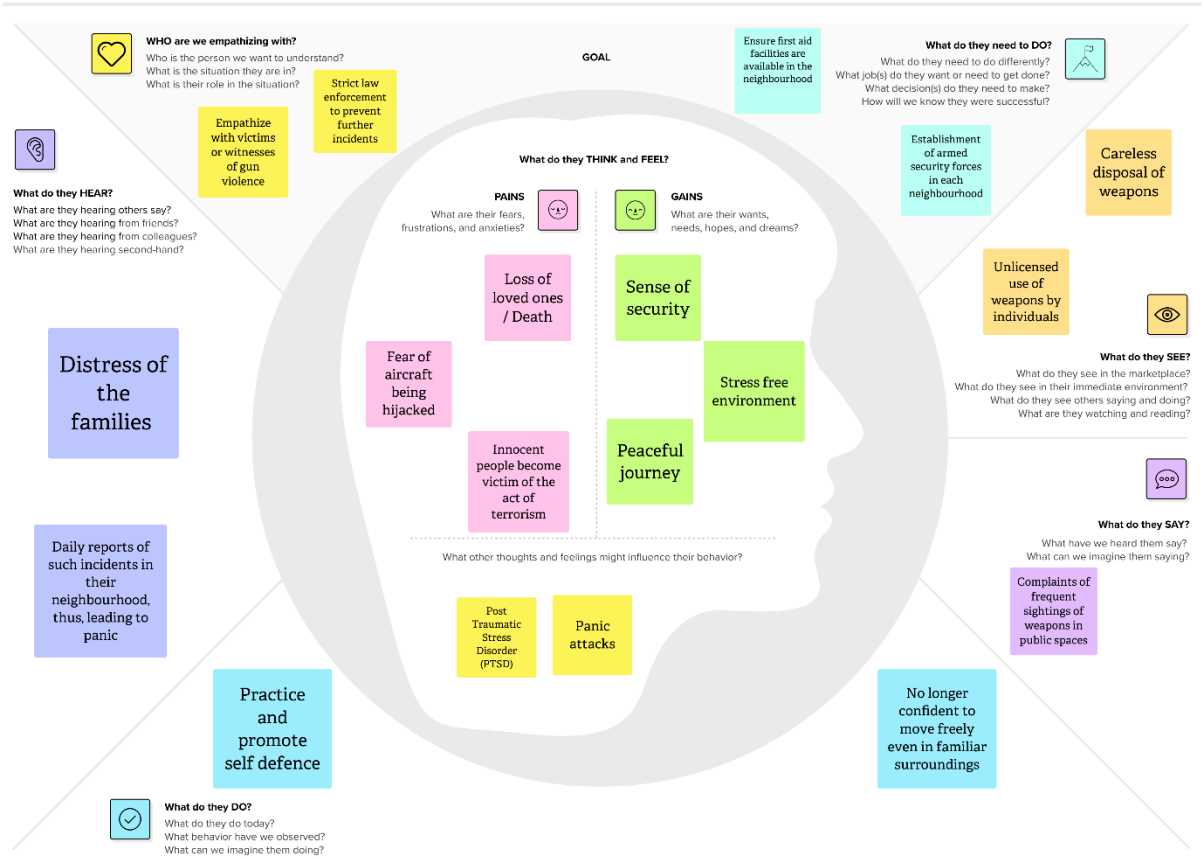
Violent incidents and public safety concerns are prevalent worldwide. Detection and prevention of weapon-related threats in public spaces, government buildings, and critical infrastructure are of paramount importance. Traditional security measures often fall short in identifying concealed or openly displayed weapons efficiently.

The project seeks to develop an accurate and real-time weapon detection system using YOLOv8 Deep Learning, which can effectively identify various types of weapons in diverse scenarios, thus enhancing security measures and reducing the potential for violent incidents.

3. Ideation & Proposed Solution

3.1 Empathy Map Canvas

To effectively address the problem of weapon detection and enhance security measures, it is essential to understand the needs and concerns of various stakeholders involved. The Empathy Map Canvas helps us gain a deeper insight into the perspectives of these stakeholders.



3.2 Ideation & Brainstorming

Proposed Solution

The proposed solution involves the development and deployment of a real-time weapon detection system using the YOLOv8 model. This solution leverages the capabilities of deep learning to address the limitations of traditional weapon detection methods.

Key Components of the Proposed Solution:

- **YOLOv8 Model**: We will use the YOLOv8 model, which represents the latest iteration of the YOLO series, known for its real-time object detection capabilities.
- **Data Collection**: We will gather a diverse dataset of images and video frames containing weapons, ensuring the representation of various scenarios.
- **Model Training**: The YOLOv8 model will be trained on the annotated dataset to learn to detect weapons accurately.
- **Real-time Detection**: The system will provide real-time detection of weapons within images and video streams.

The proposed solution aims to enhance security measures while respecting privacy and civil liberties. It will provide a scalable and efficient approach to weapon detection in public spaces, contributing to the safety of individuals and compliance with regulatory standards.

4. Requirement Analysis

4.1 Functional Requirements

Functional requirements describe the specific functions and capabilities that the weapon detection system using the YOLOv8 model must possess to meet its objectives.

4.1.1 Object Detection

The system shall accurately detect and localize weapons in images and video frames.

4.1.2 Real-time Detection

The system shall provide real-time detection capabilities, with minimal delay in processing.

4.1.3 Data Collection

The system shall support the collection of a diverse dataset of images and video frames containing weapons.

4.1.4 Model Training

The system shall facilitate the training of the YOLOv8 model on the annotated dataset.

4.1.5 Accuracy

The system shall achieve a high level of accuracy in weapon detection, minimizing false positives and false negatives.

4.1.6 Scalability

The system shall be scalable to accommodate varying deployment scenarios, from small-scale installations to large public spaces.

4.1.7 Optimization

The system shall be optimized for both accuracy and inference speed, ensuring efficient weapon detection.

4.1.8 User Interface

The user interface shall be intuitive and user-friendly, allowing easy interaction for security personnel and administrators.

4.2 Non-Functional Requirements

Non-functional requirements encompass aspects related to system performance, usability, security, and compliance.

4.2.1 Performance

- Latency: The system shall provide real-time detection with minimum possible latency.

4.2.2 Security

- The system shall implement robust security measures to protect the data and ensure system integrity.
- Access to the system settings and data shall be restricted to authorized personnel only.

4.2.3 Reliability

The system shall be highly reliable, with minimal downtime or system failures. It shall include mechanisms for system recovery in case of unexpected failures.

4.2.4 Usability

- The system shall have a user-friendly interface, facilitating ease of use for security personnel and administrators.
- It shall provide clear and informative feedback on detected weapons.

4.2.5 Portability

The system shall be designed to be portable and compatible with different deployment platforms, including cloud and edge devices.

4.2.6 Maintenance

The system shall include provisions for ongoing maintenance and updates to ensure its effectiveness over time.

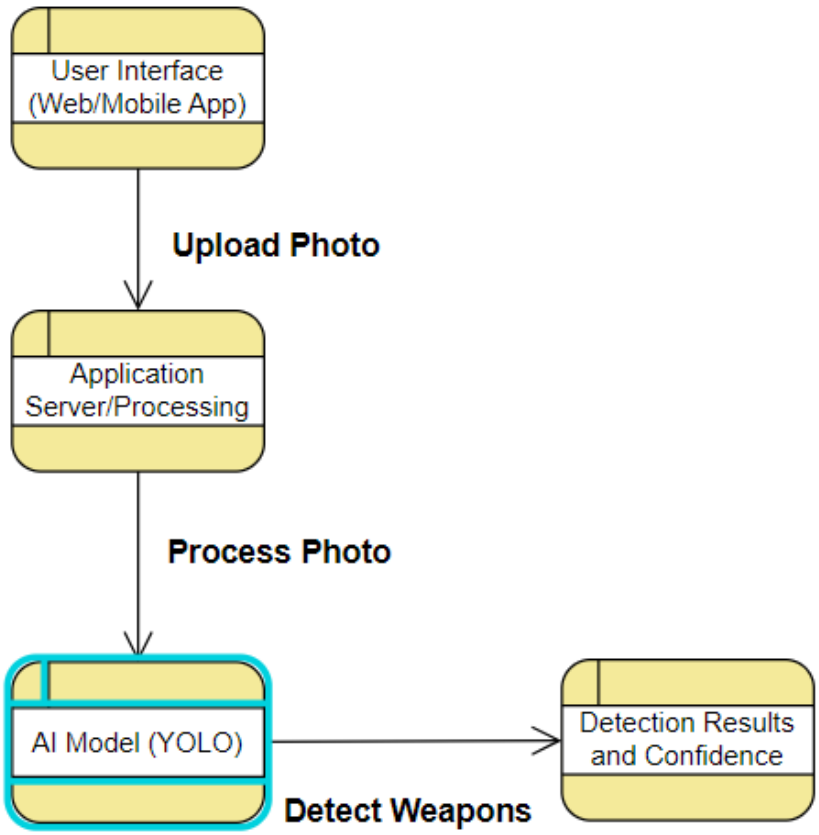
4.2.7 Documentation

Thorough documentation shall be provided to guide users and administrators on system setup, operation, and maintenance.

5. Project Design

5.1 Data Flow Diagram & User Stories

5.1.1 Data Flow Diagram



5.1.2 User stories

User stories help define the system's functionality from the perspective of different stakeholders.

Security Personnel User Story:

As a Security Officer, I want to be alerted in real-time when a weapon is detected in an image or video stream so that I can respond to potential security threats promptly.

Public User Story:

As a member of the public, I want to be assured that my privacy is protected while weapon detection is in operation, ensuring that personal data is not misused or disclosed.

Regulatory Authority User Story:

As a Regulatory Authority, I want the weapon detection system to comply with privacy and data protection regulations to ensure the system's ethical and legal use.

System Administrator User Story:

As a System Administrator, I should be able to configure and update the Weapon Detection Application. Along with it, I should be able to monitor the performance of the system.

Developers and AI Engineers User Story:

As a machine learning engineer, I want the system to be adaptable, allowing for continuous model training and updates to improve detection capabilities.

As a software developer, I want the codebase to be well-documented and maintainable, making it easy to troubleshoot and update as needed.

5.2 Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

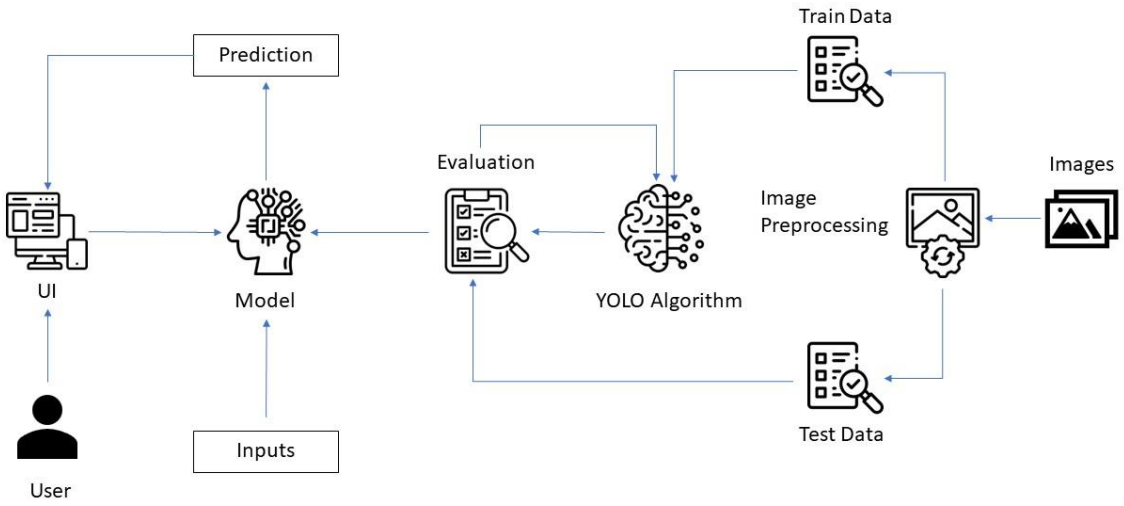
- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Our solution leverages Computer Vision (Object Detection Algorithms like YOLO) to address the weapon detection problem effectively.

YOLO's speed is a standout feature, processing entire images in a single pass through the neural network, making it ideal for applications where immediate detection and response are paramount. Moreover, YOLO offers a commendable balance of speed and accuracy, even when dealing with small or partially obscured objects, ensuring reliable performance in real-world scenarios. Its active developer community ensures ongoing improvements and adaptability to new challenges, providing longevity to the system.

- Data Collection and Annotation
- Data Preprocessing
- Model Selection and Training
- Model Evaluation
- Model Export and Deployment
- Integration with Cameras or Video Streams

5.2.1 Solution Architecture Diagram



6. Project Planning and Scheduling

6.1 Technical Architecture

The technical architecture of the weapon detection system using the YOLOv8 model is crucial for ensuring efficient and effective performance. The architecture consists of various components that work together to achieve the system's objectives.

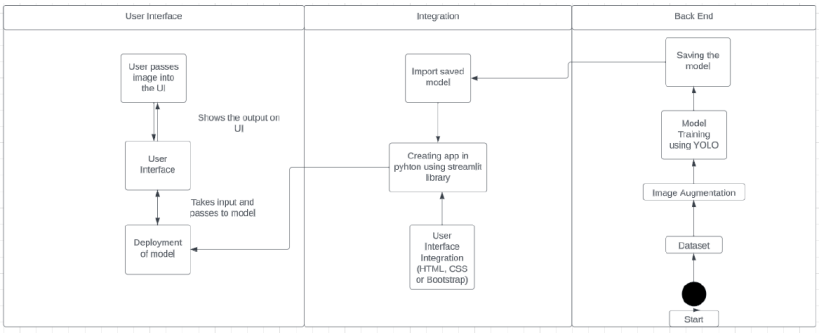
Frontend Interface: This component provides a user-friendly interface for security personnel and administrators to interact with the system.

Backend Processing: This component includes the YOLOv8 model for object detection, data collection, model training, and real-time detection.

Framework: This component integrates the frontend and backend to create a web application. The framework used in this particular project is Streamlit.

The technical architecture is designed to be modular and extensible, allowing for future updates and enhancements as the project evolves.

Technical Architecture Diagram:



6.2 Sprint Planning & Estimation

Sprint planning involves breaking down the project into smaller, manageable units of work called **Sprints**. Each sprint typically has a fixed duration and focuses on specific tasks and objectives.

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-1	Project setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the Weapon Detection project.	2	High
Sprint-1	Data collection	USN-2	Gather a diverse dataset of images containing different types of different types of weapons (handgun, sniper, rifles, etc.) for training the Object Detection learning model.	2	High
Sprint-2	Data Preprocessing	USN-3	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets.	2	High
Sprint-2	Development Environment	USN-4	Explore and evaluate different Object Detection Models (e.g., YOLOv8, YOLOv7, etc.) to select the most suitable model for Weapon Detection	3	High
Sprint-3	Model Development	USN-5	Train the selected deep learning model using the preprocessed dataset and monitor its performance on the validation set.	4	High
Sprint-3	Training	USN-6	Implement data augmentation techniques (e.g., rotation, flipping) to improve the model's robustness and accuracy.	6	Medium
Sprint-4	Model Deployment & Integration	USN-7	Deploy the trained Object Detection Model as an API or web service to make it accessible for Weapon Detection. Integrate the model's API into a user-friendly web interface for users to upload images and receive Weapon Detection results.	2	Medium
Sprint-5	Testing & Quality Assurance	USN-8	Conduct thorough testing of the model and web interface to identify and report any issues or bugs. Fine-tune the model hyperparameters and optimize its performance based on user feedback and testing results.	1	Medium

6.3 Sprint Delivery Schedule

The sprint delivery schedule outlines the planned start and end dates for each sprint, along with the anticipated delivery date for key project milestones.

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	4	3 Days	12 Oct 2023	15 Oct 2023	4	15 Oct 2023
Sprint-2	5	7 Days	16 Oct 2023	23 Oct 2023	5	23 Oct 2023
Sprint-3	10	7 Days	24 Oct 2023	31 Oct 2023	10	31 Oct 2023
Sprint-4	2	4 Days	1 Nov 2023	5 Nov 2023	2	5 Nov 2023
Sprint-5	2	3 Days	5 Nov 2023	8 Nov 2023	2	8 Nov 2023

7. Coding & Solutioning

7.1 Feature 1

Our project features the cutting-edge YOLOv8 Object Detection Model, renowned for its exceptional performance in identifying weapons in any provided image. Users can seamlessly select local image files from their devices and submit them for analysis. Thanks to a diverse and comprehensive dataset, our model excels in identifying a wide range of arms and ammunition, making it a potent tool in ensuring public safety and security. With its outstanding accuracy and swift detection capabilities, our model outperforms traditional methods, setting a new standard in weapon detection.

Code Snippet from helper.py:

```
def _display_detected_frames(conf, model, st_frame, image, is_display_tracking=None, tracker=None):
    """
    Display the detected objects on a video frame using the YOLOv8 model.

    Args:
    - conf (float): Confidence threshold for object detection.
    - model (YoloV8): A YOLOv8 object detection model.
    - st_frame (Streamlit object): A Streamlit object to display the detected video.
    - image (numpy array): A numpy array representing the video frame.
    - is_display_tracking (bool): A flag indicating whether to display object tracking (default=None).

    Returns:
    None
    """

    # Resize the image to a standard size
    image = cv2.resize(image, (720, int(720*(9/16))))

    # Display object tracking, if specified
    if is_display_tracking:
        res = model.track(image, conf=conf, persist=True, tracker=tracker)
    else:
        # Predict the objects in the image using the YOLOv8 model
        res = model.predict(image, conf=conf)

    # # Plot the detected objects on the video frame
    res_plotted = res[0].plot()
    st_frame.image(res_plotted,
                   caption='Detected Video',
                   channels="BGR",
                   use_column_width=True
                   )
```

Code Snippet from app.py:

```
source_img = None
# If image is selected
if source_radio == settings.IMAGE:
    source_img = st.sidebar.file_uploader(
        "Choose an image...", type=("jpg", "jpeg", "png", 'bmp', 'webp'))

    col1, col2 = st.columns(2)

    with col1:
        try:
            if source_img is None:
                default_image_path = str(settings.DEFAULT_IMAGE)
                default_image = PIL.Image.open(default_image_path)
                st.image(default_image_path, caption="Default Image",
                         use_column_width=True)
            else:
                uploaded_image = PIL.Image.open(source_img)
                st.image(source_img, caption="Uploaded Image",
                         use_column_width=True)
        except Exception as ex:
            st.error("Error occurred while opening the image.")
            st.error(ex)
```

```
    with col2:
        if source_img is None:
            default_detected_image_path = str(settings.DEFAULT_DETECT_IMAGE)
            default_detected_image = PIL.Image.open(
                default_detected_image_path)
            st.image(default_detected_image_path, caption='Detected Image',
                     use_column_width=True)
        else:
            if st.sidebar.button('Detect Objects'):
                res = model.predict(uploaded_image,
                                    conf=confidence
                                    )
                boxes = res[0].boxes
                res_plotted = res[0].plot()[:, :, ::-1]
                st.image(res_plotted, caption='Detected Image',
                         use_column_width=True)
                try:
                    with st.expander("Detection Results"):
                        for box in boxes:
                            st.write(box.data)
                except Exception as ex:
                    # st.write(ex)
                    st.write("No image is uploaded yet!")
```

7.2 Feature 2

Our model breaks free from the confines of static imagery and embraces the dynamic world of videos. By seamlessly integrating our state-of-the-art YOLOv8 model with the powerful OpenCV library, we empower our system to go beyond mere image analysis. It offers an exciting and dynamic capability — the incorporation of videos for detection and analysis.

Whether it is a routine check or a critical investigation by any security personnel, our model transforms video playback into a valuable tool for enhancing security. With its ability to process videos, our system opens up new horizons in surveillance, giving security experts the power to analyse historical footage and uncover essential insights.

Code Snippet from helper.py:

```python
def play_stored_video(conf, model):
    """
    Plays a stored video file. Tracks and detects objects in real-time using the YOLOv8 object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_vid = st.sidebar.selectbox(
        "Choose a video...", settings.VIDEOS_DICT.keys())

    is_display_tracker, tracker = display_tracker_options()

    with open(settings.VIDEOS_DICT.get(source_vid), 'rb') as video_file:
        video_bytes = video_file.read()
    if video_bytes:
        st.video(video_bytes)
```

```python
    if st.sidebar.button('Detect Video Objects'):
        try:
            vid_cap = cv2.VideoCapture(
                str(settings.VIDEOS_DICT.get(source_vid)))
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
                                             is_display_tracker,
                                             tracker
                                             )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))
```

7.3 Feature 3

At the heart of our system lies a standout and unique feature — the capability for real-time weapon detection through live streaming, seamlessly facilitated by webcams. Users can effortlessly connect their application to their system's cameras, instantly converting their environment into a vigilant, real-time surveillance platform.

Code Snippet from helper.py:

```
def play_webcam(conf, model):
    """
    Plays a webcam stream. Detects Objects in real-time using the YOLOv8 object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_webcam = settings.WEBCAM_PATH
    is_display_tracker, tracker = display_tracker_options()
    if st.sidebar.button('Detect Objects'):
        try:
            vid_cap = cv2.VideoCapture(source_webcam)
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
                                             is_display_tracker,
                                             tracker,
                                             )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))
```

7.4 Feature 4

The application features a user-friendly interface designed for ease of use. The interface enhances the overall user experience and simplifies the management of weapon detection tasks. The user interface is intuitive and straightforward, making it accessible to security personnel with minimal training. It offers clear navigation, allowing users to easily access different functionalities, including image and video upload. The results of weapon detection are presented in a clear and understandable manner. The system highlights the location of detected weapons. The interface supports real-time monitoring and highlights threats by drawing boxes around the weapon during live streaming.

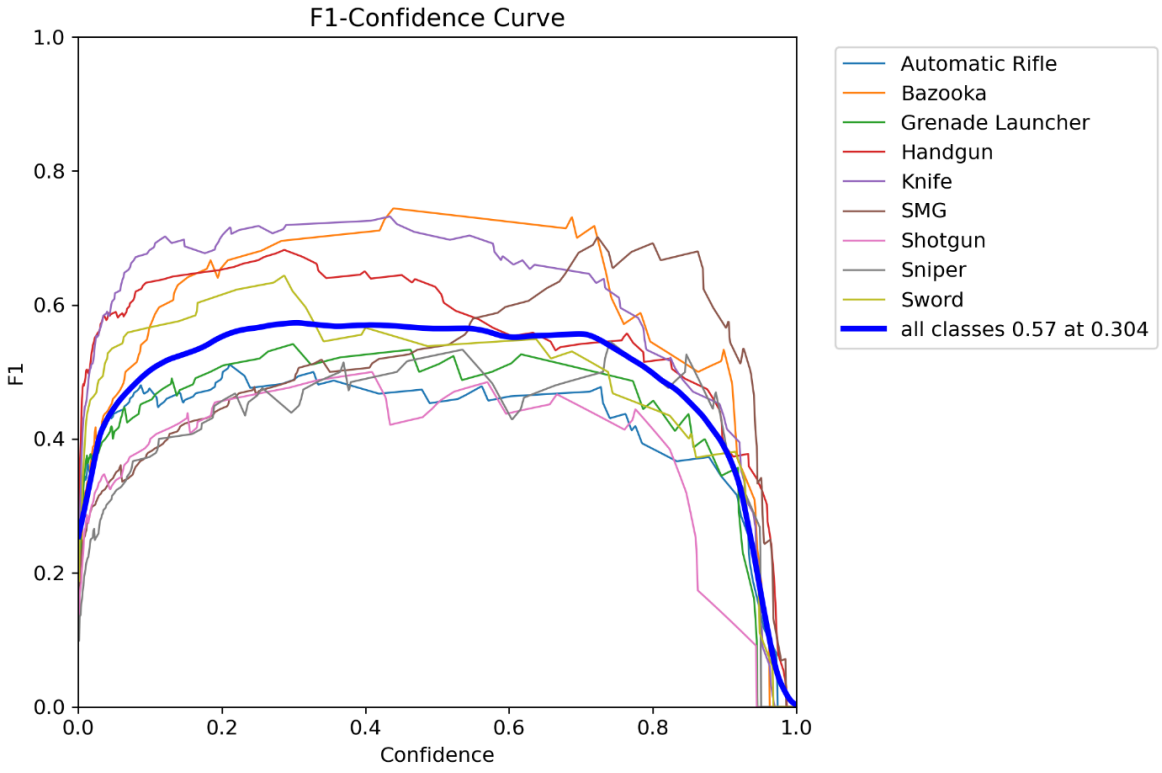
8. Performance Testing

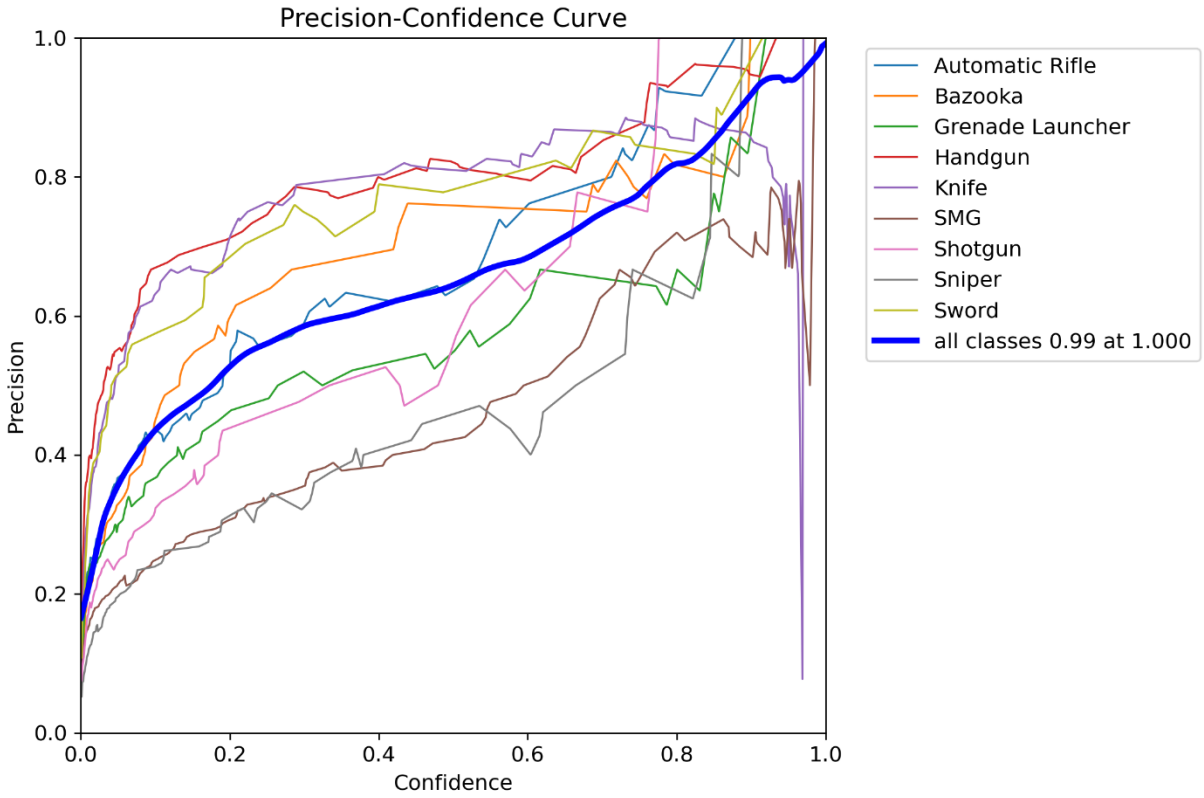
8.1 Performance Metrics

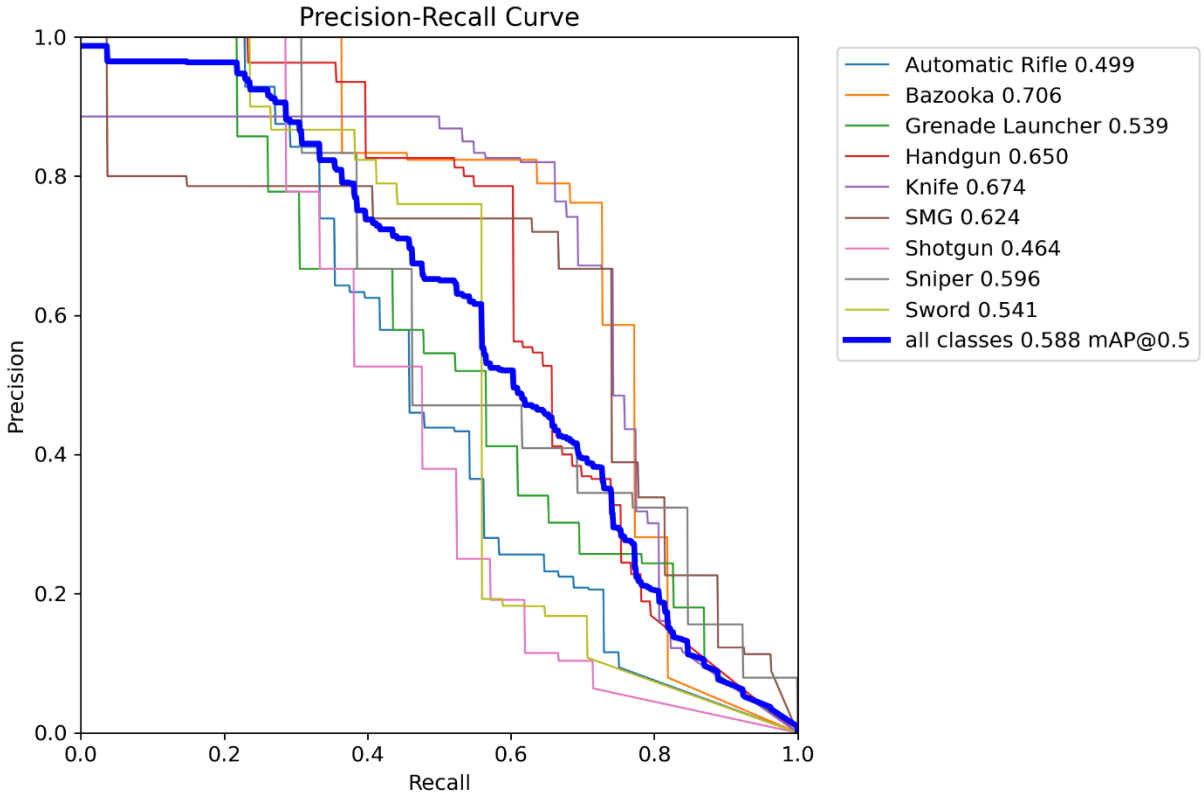
8.1.1 Model Summary

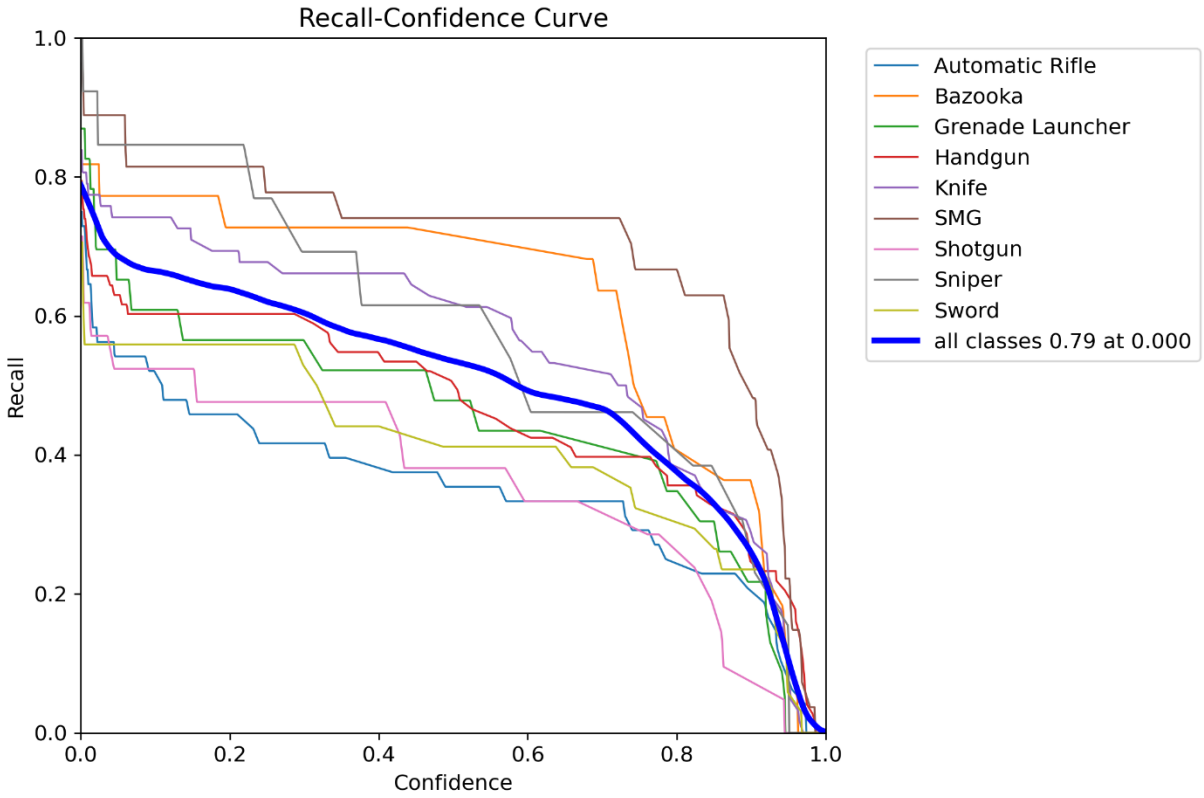
```
Model summary (fused): 268 layers, 43613547 parameters, 0 gradients, 164.9 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 7/7 [00:08<00:00,  1.20s/it]
                   all        210        323      0.588      0.602      0.588      0.427
       Automatic Rifle        210         48      0.603      0.417      0.499      0.328
               Bazooka        210         22      0.671      0.727      0.706      0.493
     Grenade Launcher        210         23      0.515      0.554      0.539      0.398
               Handgun        210         73      0.783      0.593       0.65      0.486
                 Knife        210         62      0.791      0.661      0.674      0.455
                   SMG        210         27       0.37      0.778      0.624      0.513
               Shotgun        210         21      0.484      0.476      0.464      0.321
                Sniper        210         13      0.331      0.692      0.596      0.436
                 Sword        210         34      0.746      0.518      0.541      0.412
Speed: 0.3ms preprocess, 16.6ms inference, 0.0ms loss, 5.7ms postprocess per image
```

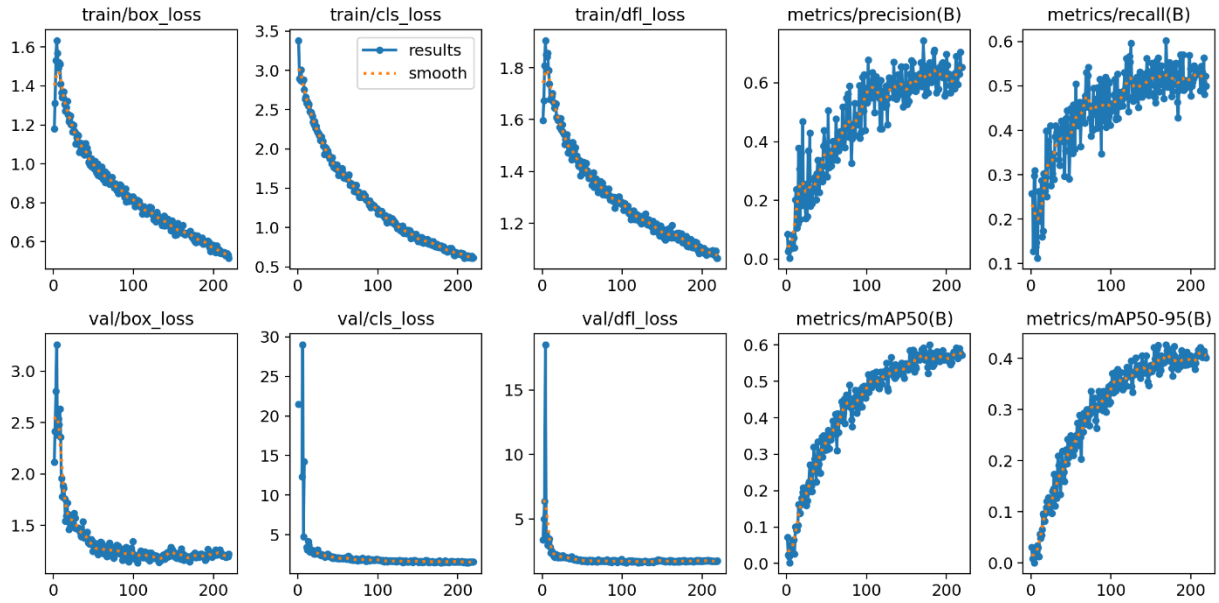
8.1.2 Confidence Scores



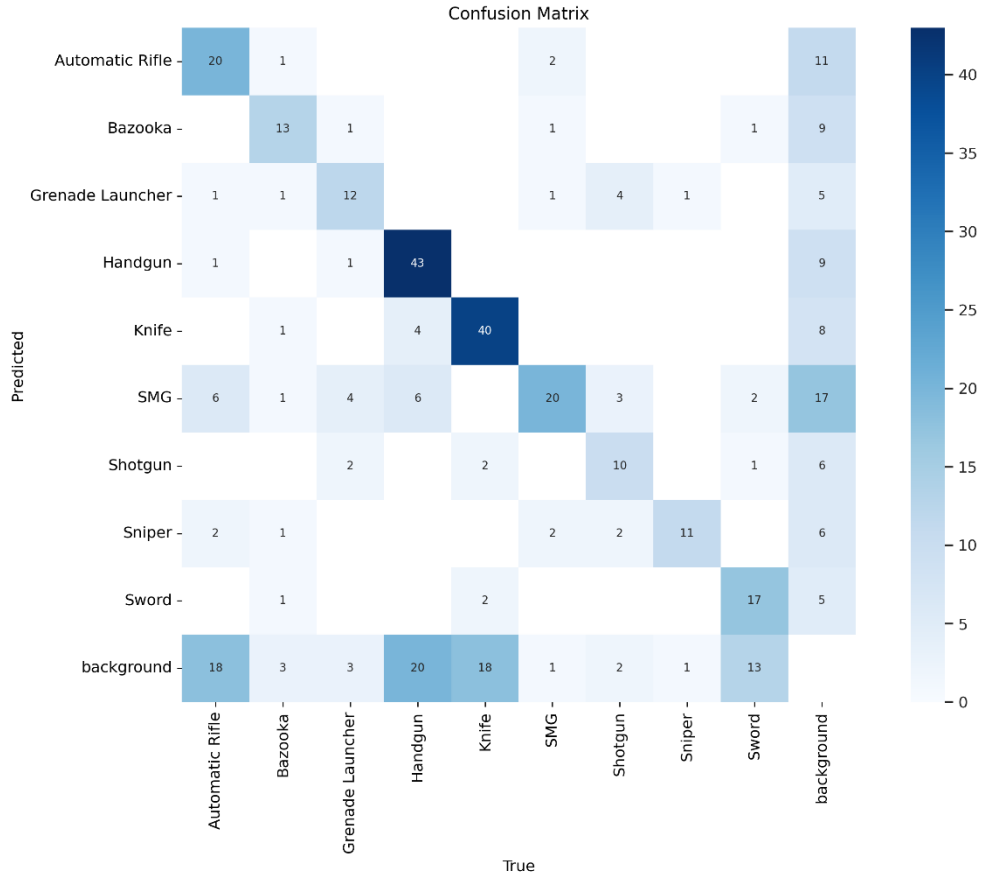

Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Automatic Rifle
Bazooka
Grenade Launcher
Handgun
Knife
SMG
Shotgun
Sniper
Sword
all classes 0.99 at 1.000


Precision-Recall Curve
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
Automatic Rifle 0.499
Bazooka 0.706
Grenade Launcher 0.539
Handgun 0.650
Knife 0.674
SMG 0.624
Shotgun 0.464
Sniper 0.596
Sword 0.541
all classes 0.588 mAP@0.5


Recall-Confidence Curve
Recall
Confidence
Automatic Rifle
Bazooka
Grenade Launcher
Handgun
Knife
SMG
Shotgun
Sniper
Sword
all classes 0.79 at 0.000


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

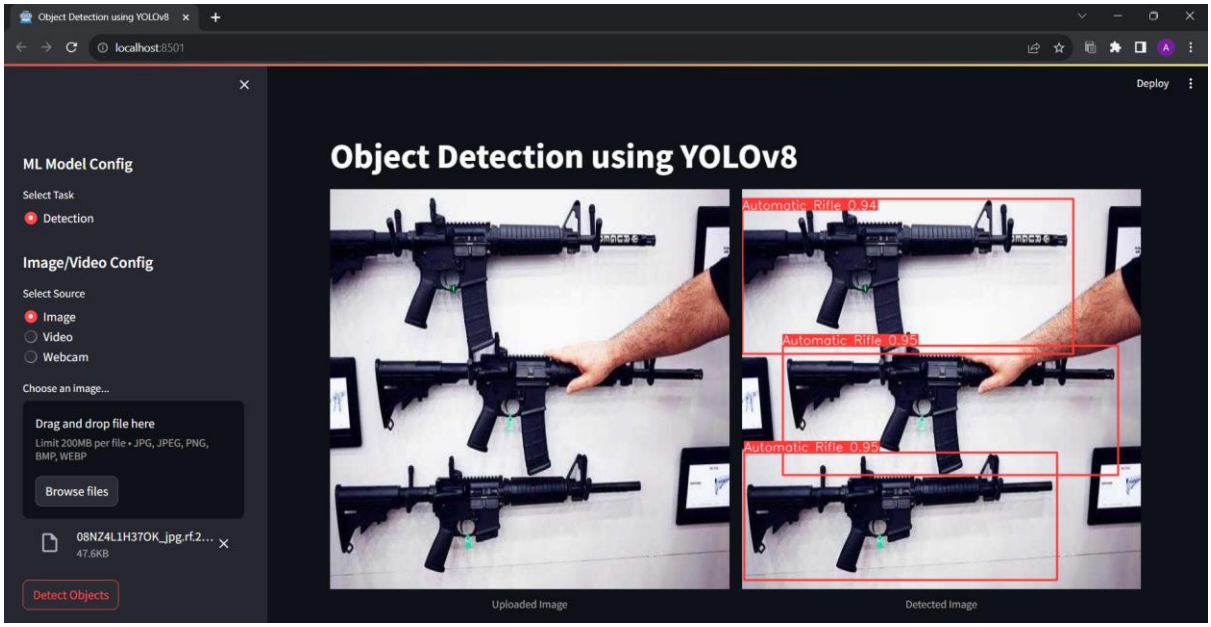
8.1.3 Confusion Matrix



9. Results

9.1 Output Screenshots

9.1.1 Image as an Input

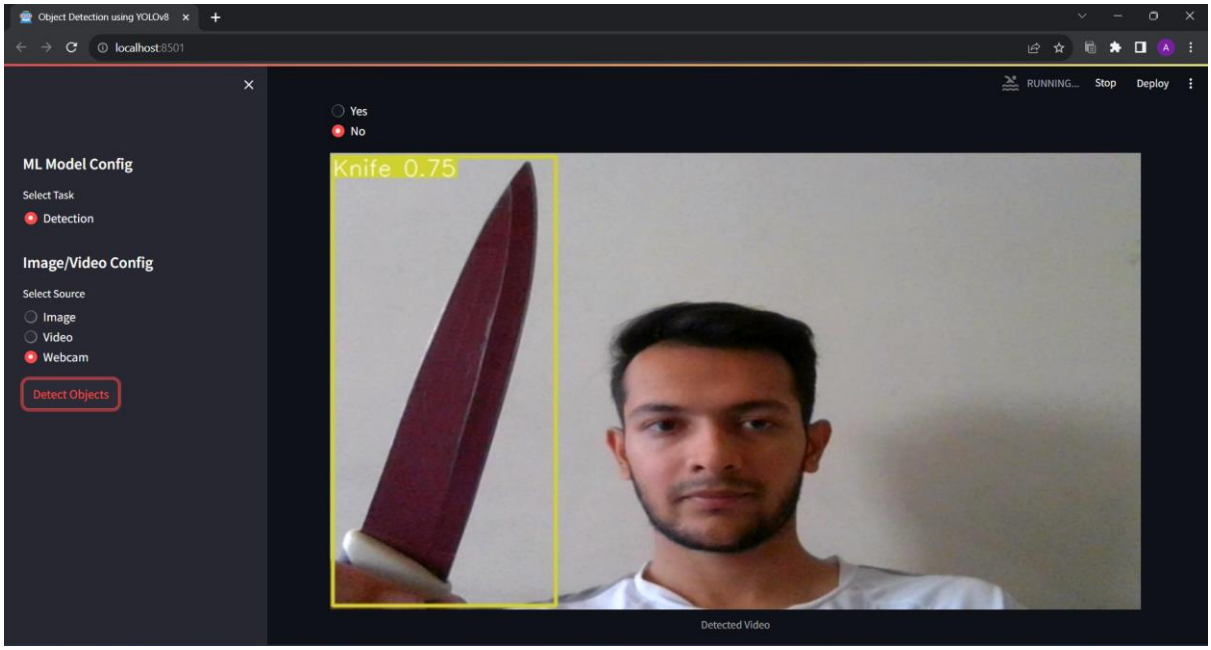


9.1.2 Video as an Input

Please double-click on the file below to open.

Object Detection using YOLOv8 - Google Chrome 2023-11-09 09-56-55.mp4

9.1.3 Live Stream Using Webcam



10. Advantages & Disadvantages

10.1 Advantages of the Proposed Solution

1. Accurate Weapon Detection: The integration of the YOLOv8 model allows for precise weapon detection, enabling the system to identify and locate weapons within uploaded images accurately.

2. Efficiency: YOLOv8's architecture is designed for efficiency, allowing it to process images and videos with remarkable speed and accuracy. This efficiency extends to its memory and computational requirements, making it feasible for deployment on a variety of hardware platforms.

3. Real-Time Detection: The project provides real-time weapon detection capabilities, allowing for immediate responses to potential security threats. This real-time aspect is essential in enhancing public safety and security.

4. User-Friendly Interface: The frontend, built using Streamlit, provides a user-friendly interface for uploading images or videos, and viewing the detection results. The intuitive design enhances the user experience and ease of navigation.

5. Versatility: Our project is not limited to a single type of weapon; it can detect various classes of arms and ammunition. This versatility makes it applicable in diverse security scenarios.

6. Ethical Considerations: Your project addresses ethical concerns, including the reduction of biases in detection outcomes and the implementation of privacy protection measures. This responsible approach aligns with ethical standards in AI and security applications.

10.2 Disadvantages of the Proposed Solution

1. Dependency on Image Quality: The accuracy of weapon detection and subsequent processing heavily relies on the quality of the uploaded images. Low-resolution or distorted images may result in inaccurate ingredient identification and classification.

2. False Positives and False Negatives: Despite high accuracy, no system is perfect, and there may still be instances of false positives (incorrectly identifying an object as a weapon) and false negatives (failing to detect a weapon).

3. Environmental Limitations: Certain environmental conditions, such as extreme weather, poor lighting, or challenging physical surroundings, may pose difficulties for surveillance and detection systems.

4. Data Storage and Retention: Storing large volumes of video footage and detection data can be resource-intensive and may require dedicated storage solutions.

5. Hardware and Software Requirements: The project's high-performance requirements may necessitate powerful hardware, which could be costly to implement. Additionally, ensuring compatibility with different camera systems and software platforms may be challenging.

It is important to note that some of these disadvantages can be mitigated through continuous model training, data augmentation, and improvements in the underlying algorithms and techniques employed in the solution.

11. Conclusion

Throughout the development and implementation of the project, several key findings and achievements have been made:

1. Accurate Weapon Detection: The integration of the YOLOv8 model allows for precise weapon detection, enabling the system to identify and locate weapons within uploaded images accurately.

2. Efficiency: YOLOv8's architecture is designed for efficiency, allowing it to process images and videos with remarkable speed and accuracy. This efficiency extends to its memory and computational requirements, making it feasible for deployment on a variety of hardware platforms.

3. User-Friendly Interface: The frontend, built using Streamlit, provides a user-friendly interface for uploading images or videos, and viewing the detection results. The intuitive design enhances the user experience and ease of navigation.

The system's real-time weapon detection with live streaming through webcams presents a unique and essential feature, providing security personnel with immediate insights into potential threats. This feature not only enhances security operations but also contributes to the prevention of crimes by deterring individuals from carrying weapons in monitored areas.

While the proposed solution demonstrates promising results, it is essential to acknowledge some limitations. The accuracy of weapon detection may depend on image quality, and various environmental conditions can impact weapon recognition.

Moving forward, further enhancements can be made to address these limitations and improve the solution's overall performance. Continual model training, data augmentation, and advancements in AI and ML techniques can contribute to the system's robustness and accuracy.

Overall, our YOLOv8-based weapon detection project represents a powerful tool for improving public safety, streamlining security operations, and responding rapidly to security breaches. Its robustness, versatility, and real-time capabilities make it a valuable addition to the field of security and surveillance.

12. Future Scope

The proposed project lays the groundwork for further advancements and improvements in AI-driven weapon detection systems. Here are some future scope and potential enhancements that can be considered:

1. Enhanced Accuracy: Continued research and development can further improve the accuracy of the weapon detection system, reducing false positives and false negatives. Implementing advanced machine learning techniques and refining the training data can contribute to this.

2. Continuous Model Training: Regularly updating and retraining the object detection models with new data can improve their accuracy and keep them up-to-date with evolving culinary trends and ingredient variations.

3. Expansion of Weapon Recognition: The system can be further developed to recognize a wider range of weapons, including special army grade machinery. This can be achieved by augmenting the dataset, training the models on a more diverse set of weapons, and exploring transfer learning techniques.

4. Integration with Website Hosting Services: Currently, Streamlit, the platform we used to deploy and host our model, does not support the access of webcams on their Cloud Servers. Further updates on the platform will ensure maximum utilisation of the weapon detection model.

These future scope and enhancements can further enhance the functionality, accuracy, and user experience of the system, making it a more comprehensive and valuable tool for weapon detection and surveillance.

13. Appendix

Source Code:

1. settings.py

```
from pathlib import Path
import sys

# Get the absolute path of the current file
file_path = Path(__file__).resolve()

# Get the parent directory of the current file
root_path = file_path.parent

# Add the root path to the sys.path list if it is not already there
if root_path not in sys.path:
    sys.path.append(str(root_path))

# Get the relative path of the root directory with respect to the current
working directory
ROOT = root_path.relative_to(Path.cwd())

# Sources
IMAGE = 'Image'
VIDEO = 'Video'
WEBCAM = 'Webcam'

SOURCES_LIST = [IMAGE, VIDEO, WEBCAM]

# Images config
IMAGES_DIR = ROOT / 'test' / 'images'
DEFAULT_IMAGE = IMAGES_DIR / 'weapons2.jpg'
DEFAULT_DETECT_IMAGE = IMAGES_DIR / 'weapons2Detected.jpg'

# Videos config
VIDEO_DIR = ROOT / 'test' / 'videos'
VIDEO_1_PATH = VIDEO_DIR / 'beyonce-gun.gif'
VIDEO_2_PATH = VIDEO_DIR / 'knife-deep-thinking.gif'
VIDEO_3_PATH = VIDEO_DIR / 'gun-pistol.gif'
VIDEO_4_PATH = VIDEO_DIR / 'machine-gun-498-x-247-gif-jdurh7cgwvbh401x.gif'
VIDEO_5_PATH = VIDEO_DIR / 'ITna.gif'
VIDEO_6_PATH = VIDEO_DIR / 'tumblr_mus2ooy2iX1sl27r8o1_500.gif'
VIDEO_7_PATH = VIDEO_DIR / '4cfc146df61c2df780e1bb8bd3744c9a.gif'
VIDEO_8_PATH = VIDEO_DIR / 'e4fb81f935a0fec378f82c7b5c0173c6.gif'
VIDEO_9_PATH = VIDEO_DIR / 'sniper.gif'
VIDEOS_DICT = {
    'Video 1': VIDEO_1_PATH,
```

```
    'Video 2': VIDEO_2_PATH,
    'Video 3': VIDEO_3_PATH,
    'Video 4': VIDEO_4_PATH,
    'Video 5': VIDEO_5_PATH,
    'Video 6': VIDEO_6_PATH,
    'Video 7': VIDEO_7_PATH,
    'Video 8': VIDEO_8_PATH,
    'Video 9': VIDEO_9_PATH,
}

# ML Model config
MODEL_DIR = ROOT / 'runs'/'detect' / 'train' / 'weights'
DETECTION_MODEL = MODEL_DIR / 'best.pt'

# Webcam
WEBCAM_PATH = 0
```

2. helper.py

```
from ultralytics import YOLO
import time
import streamlit as st
import cv2

import settings


def load_model(model_path):
    """
    Loads a YOLO object detection model from the specified model_path.

    Parameters:
        model_path (str): The path to the YOLO model file.

    Returns:
        A YOLO object detection model.
    """
    model = YOLO(model_path)
    return model


def display_tracker_options():
    display_tracker = st.radio("Display Tracker", ('Yes', 'No'))
    is_display_tracker = True if display_tracker == 'Yes' else False
    if is_display_tracker:
        tracker_type = st.radio("Tracker", ("bytetrack.yaml", "botsort.yaml"))
        return is_display_tracker, tracker_type
    return is_display_tracker, None


def _display_detected_frames(conf, model, st_frame, image,
is_display_tracking=None, tracker=None):
    """
    Display the detected objects on a video frame using the YOLOv8 model.

    Args:
    - conf (float): Confidence threshold for object detection.
    - model (YoloV8): A YOLOv8 object detection model.
    - st_frame (Streamlit object): A Streamlit object to display the detected
video.
    - image (numpy array): A numpy array representing the video frame.
    - is_display_tracking (bool): A flag indicating whether to display object
tracking (default=None).

    Returns:
    None
```

```
    """

    # Resize the image to a standard size
    image = cv2.resize(image, (720, int(720*(9/16))))

    # Display object tracking, if specified
    if is_display_tracking:
        res = model.track(image, conf=conf, persist=True, tracker=tracker)
    else:
        # Predict the objects in the image using the YOLOv8 model
        res = model.predict(image, conf=conf)

    # # Plot the detected objects on the video frame
    res_plotted = res[0].plot()
    st_frame.image(res_plotted,
                   caption='Detected Video',
                   channels="BGR",
                   use_column_width=True
                   )


def play_webcam(conf, model):
    """
    Plays a webcam stream. Detects Objects in real-time using the YOLOv8
object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_webcam = settings.WEBCAM_PATH
    is_display_tracker, tracker = display_tracker_options()
    if st.sidebar.button('Detect Objects'):
        try:
            vid_cap = cv2.VideoCapture(source_webcam)
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
```

```
                                            is_display_tracker,
                                            tracker,
                                            )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))


def play_stored_video(conf, model):
    """
    Plays a stored video file. Tracks and detects objects in real-time using
the YOLOv8 object detection model.

    Parameters:
        conf: Confidence of YOLOv8 model.
        model: An instance of the `YOLOv8` class containing the YOLOv8 model.

    Returns:
        None

    Raises:
        None
    """
    source_vid = st.sidebar.selectbox(
        "Choose a video...", settings.VIDEOS_DICT.keys())

    is_display_tracker, tracker = display_tracker_options()

    with open(settings.VIDEOS_DICT.get(source_vid), 'rb') as video_file:
        video_bytes = video_file.read()
    if video_bytes:
        st.video(video_bytes)

    if st.sidebar.button('Detect Video Objects'):
        try:
            vid_cap = cv2.VideoCapture(
                str(settings.VIDEOS_DICT.get(source_vid)))
            st_frame = st.empty()
            while (vid_cap.isOpened()):
                success, image = vid_cap.read()
                if success:
                    _display_detected_frames(conf,
                                             model,
                                             st_frame,
                                             image,
                                             is_display_tracker,
                                             tracker
```

```
                                            )
                else:
                    vid_cap.release()
                    break
        except Exception as e:
            st.sidebar.error("Error loading video: " + str(e))
```

3. app.py

```python
# Python In-built packages
from pathlib import Path
import PIL

# External packages
import streamlit as st

# Local Modules
import settings
import helper

# Setting page layout
st.set_page_config(
    page_title="Object Detection using YOLOv8",
    page_icon="🤖",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Main page heading
st.title("Object Detection using YOLOv8")

# Sidebar
st.sidebar.header("ML Model Config")

# Model Options
model_type = st.sidebar.radio(
    "Select Task", ['Detection'])

confidence = 0.25

# Selecting Detection Or Segmentation
if model_type == 'Detection':
    model_path = Path(settings.DETECTION_MODEL)

# Load Pre-trained ML Model
try:
    model = helper.load_model(model_path)
except Exception as ex:
    st.error(f"Unable to load model. Check the specified path: {model_path}")
    st.error(ex)

st.sidebar.header("Image/Video Config")
source_radio = st.sidebar.radio(
    "Select Source", settings.SOURCES_LIST)

source_img = None
```

```
# If image is selected
if source_radio == settings.IMAGE:
    source_img = st.sidebar.file_uploader(
        "Choose an image...", type=("jpg", "jpeg", "png", 'bmp', 'webp'))

    col1, col2 = st.columns(2)

    with col1:
        try:
            if source_img is None:
                default_image_path = str(settings.DEFAULT_IMAGE)
                default_image = PIL.Image.open(default_image_path)
                st.image(default_image_path, caption="Default Image",
                         use_column_width=True)
            else:
                uploaded_image = PIL.Image.open(source_img)
                st.image(source_img, caption="Uploaded Image",
                         use_column_width=True)
        except Exception as ex:
            st.error("Error occurred while opening the image.")
            st.error(ex)

    with col2:
        if source_img is None:
            default_detected_image_path = str(settings.DEFAULT_DETECT_IMAGE)
            default_detected_image = PIL.Image.open(
                default_detected_image_path)
            st.image(default_detected_image_path, caption='Detected Image',
                     use_column_width=True)
        else:
            if st.sidebar.button('Detect Objects'):
                res = model.predict(uploaded_image,
                                    conf=confidence
                                    )
                boxes = res[0].boxes
                res_plotted = res[0].plot()[:, :, ::-1]
                st.image(res_plotted, caption='Detected Image',
                         use_column_width=True)
                try:
                    with st.expander("Detection Results"):
                        for box in boxes:
                            st.write(box.data)
                except Exception as ex:
                    # st.write(ex)
                    st.write("No image is uploaded yet!")

elif source_radio == settings.VIDEO:
    helper.play_stored_video(confidence, model)
```

```
elif source_radio == settings.WEBCAM:
    helper.play_webcam(confidence, model)

else:
    st.error("Please select a valid source type!")
```

GitHub & Project Demo Link:

GitHub Link: https://github.com/smartinternz02/SI-GuidedProject-603486-1697649792

Project Demo Link:
https://drive.google.com/file/d/1nON14dy_UsLiAet5_ya90pdol7PruFeG/view?usp=sharing