AYDIN ADNAN MENDERES UNIVERSITY

ENGINEERING FACULTY
COMPUTER ENGINEERING DEPARTMENT

DETECTION OF HAZARDOUS SITUATIONS/EQUIPMENTS

Ali Cemal GÜLMEZ

Cemal KADIOĞLU

Engin Halil YEDİRMEZ

Supervisor:

Dr. Samsun M. BAŞARICI

2025

ABSTRACT

Detection of Hazardous Situations/Equipments

Ali Cemal GÜLMEZ
Cemal KADIOĞLU
Engin Halil YEDİRMEZ

B.Sc. Thesis, Computer Engineering Department

Supervisor: Dr. Samsun M. BAŞARICI

2025, 30 pages

With increasing global security risks, the need for threat detection in public areas has become essential. Detecting weapons such as guns and knives in real-time can help prevent potential incidents, protect civilians, and support law enforcement units. Artificial intelligence (AI) and computer vision provide scalable solutions for such tasks.

This thesis focuses on the development of a weapon detection system using the YOLOv10m model. While previous works and initial stages used YOLOv8, this version adopts the most recent YOLOv10 architecture, which brings improvements in accuracy, speed, and model size.

The system is trained to detect two main weapon classes — guns and knives — in both images and video frames. The proposed system is capable of detection using a custom-trained deep learning model, visualizing predictions with bounding boxes, and running on a browser-based interface for end-user accessibility.

In the earlier phase of the project (Fall semester), a similar architecture was tested using YOLOv8 to prove feasibility. Lessons learned from that implementation informed the improved system presented in this final thesis.

Keywords: YOLOv10m, Object Detection, Weapon Detection, Deep Learning, Surveillance

ÖZET

Tehlikeli Nesne ve Durum Tespiti

Ali Cemal GÜLMEZ
Cemal KADIOĞLU
Engin Halil YEDİRMEZ

Lisans Bitirme Tezi, Bilgisayar Mühendisliği Bölümü

Danışman: Samsun M. BAŞARICI

2025, 30 sayfa

Artan küresel güvenlik riskleriyle birlikte, kamuya açık alanlarda tehdit algılama ihtiyacı elzem hale gelmiştir. Silah ve bıçak gibi silahların gerçek zamanlı olarak tespit edilmesi, olası olayların önlenmesine, sivillerin korunmasına ve kolluk birimlerinin desteklenmesine yardımcı olabilir. Yapay zeka (AI) ve bilgisayarla görme bu tür görevler için ölçeklenebilir çözümler sunmaktadır.

Bu tez, YOLOv10m modelini kullanarak bir silah tespit sisteminin geliştirilmesine odaklanmaktadır. Önceki çalışmalarda ve ilk aşamalarda YOLOv8 kullanılırken, bu sürümde doğruluk, hız ve model boyutunda iyileştirmeler getiren yeni bir mimari olan YOLOv10 mimarisi benimsenmiştir.

Sistem, hem görüntülerde hem de video karelerinde iki ana silah sınıfını (silahlar ve bıçaklar) tespit etmek üzere eğitilmiştir. Bu sayede özel olarak eğitilmiş bir derin öğrenme modeli kullanarak tespit yapabiliyor, tahminleri sınırlayıcı kutularla görselleştirebiliyor ve son kullanıcı erişilebilirliği için tarayıcı tabanlı bir arayüz üzerinde çalışabiliyor.

Projenin önceki aşamasında (Güz dönemi), benzer bir mimari YOLOv8 kullanılarak test edilmiştir. Bu uygulamadan alınan dersler, bu nihai tezde sunulan geliştirilmiş sistemin oluşmasında katkıda bulundu.

Anahtar Kelimeler: YOLOv10m, Nesne Algılama, Silah Algılama, Derin Öğrenme

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our thesis advisor, Dr. Samsun M. BAŞARICI, for his valuable guidance, encouragement, and insightful feedback throughout the project. His support was crucial in shaping the structure and direction of our study.

We would also like to thank our peers and colleagues for their moral support and constructive ideas during the implementation phase. Additionally, I would like to acknowledge the importance of open-source libraries and tools, such as YOLO, OpenCV and Python; which played a crucial role in facilitating the implementation of machine learning models and data processing techniques.

Lastly, we appreciate the infrastructure provided by Google Colab Pro, which allowed us to carry out efficient training on high-performance GPUs.

This project would not have been possible without the collaborative effort and team spirit among us.

TABLE OF CONTENTS:

1. INTRODUCTION

1.1 Reasoning

In recent years, the need for automatic threat detection has significantly increased due to rising global safety concerns. Among various computer vision applications, weapon detection has become a crucial focus area, especially for public security monitoring systems. The integration of real-time object detection algorithms with surveillance cameras can help detect potentially dangerous items and alert authorities immediately.

This thesis aims to develop a lightweight, fast, and accurate weapon detection system using the YOLOv10m architecture. YOLO (You Only Look Once) models are widely adopted for their real-time detection capabilities and efficiency. The YOLOv10 family brings further improvements in speed and accuracy compared to its predecessors.

In this study, we trained a model to detect two specific classes of weapons: knives and guns. The model was trained using a labeled dataset obtained from Roboflow, enhanced with data augmentation techniques, and deployed in a browser-based application for real-world testing. This approach offers a practical solution for smart security systems that require minimal latency and maximum reliability.

1.2 Aim

The primary goal of this project is to develop a weapon detection system using YOLOv10. This system will analyze images and videos to detect the presence of weapons (such as guns and knives) with high accuracy.

1.3 Key Objectives

1. Reduce Security Risks: The idea of the model can be improved and integrated into CCTV cameras, security systems, or mobile applications to identify weapons in real time. This could help prevent violent incidents in public places like airports, schools, shopping malls, and stadiums.
2. Automate Weapon Detection for Law Enforcement & Security: Instead of manual monitoring, security personnel can use AI-powered surveillance to quickly detect threats.
3. Fast & Accurate Detection: Uses YOLOv8, which is optimized for speed and accuracy, allowing for instant detection. The model is trained with custom weapon datasets to improve precision.
4. Deployable for Different Use Cases

1.4 How It Works

The system trains YOLOv10 on a dataset of weapon images.

It detects weapons in new images or videos. Marks detected weapons with bounding boxes. Integrates with a web-based interface. Shows the detection results in the website. Can be integrated into real-time security systems.

1.5 Potential Applications

- Airport & Public Transportation Security – Detect weapons before threats escalate.
- School Safety – Prevent incidents by monitoring entrances.
- Retail & Banking Security – Detect suspicious activities.
- Smart Cities & Law Enforcement – AI-powered surveillance for crime prevention.

2. MATERIALS & METHODS

2.1 Materials

2.1.1 Computers

High performance computers with advanced GPUs were utilized for processing the datasets and running the learning models. The computers also equipped tith sufficient RAM and storage to handle the datasets and tasks.

2.1.2 Cameras (for Real-Time Detection if needed)

As the project begins to develop towards the real-time monitoring (RTM) phase, real-time camera footage will help to obtain the necessary data set for the detection of weapons or other hazardous materials.

2.1.3 Software & Tools

Python	Programming language for machine learning & deep learning
OpenCV	Used for image processing and visualization
YOLO (ultralytics)	Object detection model used to identify hazardous objects like knives and guns (YOLOv10)
Javascript	Used for designing user-friendly web interface
HTML & CSS	Used for designing user-friendly web interface
Google Colab	Cloud-based training if local GPU is not available
Roboflow	Finding, preprocessing and augmenting the dataset

Table 1: Softwares & tools applied

2.1.3.1 Python

Python is a high-level, interpreted programming language known for its simplicity, readability, and vast ecosystem of libraries. It is widely used across fields such as web development, data science, artificial intelligence, automation, and more.

Python emphasizes code readability and allows developers to express concepts in fewer lines of code compared to many other languages.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

Python features a large standard library and integrates seamlessly with external packages via tools like “pip”, enabling rapid development in areas such as machine learning, image processing, and data visualization.

Python’s interpreted nature allows for dynamic typing and interactive development, making it an ideal language for both beginners and advanced users.

Python is platform-independent and supported on major operating systems such as Windows, Linux, and macOS.

Python has a large and active community, which contributes to thousands of third-party libraries like NumPy (numerical computing), Pandas (data analysis), OpenCV (image processing), and TensorFlow/PyTorch (deep learning).

Python is widely used in academia and industry, powering everything from simple scripts to complex machine learning systems and web applications.

2.1.3.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning library that provides a wide range of tools and functions for image and video processing, helping developers efficiently perform complex tasks.

Initially developed by Intel, OpenCV is written in C++ but supports languages like Python, Java, and MATLAB. It is widely used in robotics, medical imaging, and video surveillance.

OpenCV reads, writes, and manipulates images (e.g., resizing, rotating, filtering) and performs tasks like edge detection, color space conversion, and thresholding.

It also processes video streams from files, webcams, or cameras, handling tasks like background subtraction, motion detection, and video stabilization.

It detects and matches keypoints in images using algorithms like SIFT, SURF, and ORB, useful for object recognition, panorama stitching, and SLAM.

OpenCV detects objects using Haar cascades, HOG descriptors, or pre-trained deep learning models.

It tracks objects in video streams using algorithms like KCF, GOTURN, and MedianFlow.

OpenCV can segment an image using techniques like watershed, graph-based segmentation, and contour detection.

It provides tools for stereo vision, 3D reconstruction, and depth estimation. Its built-in machine learning module supports classification, regression, and clustering, including SVM, k-Nearest Neighbors, Decision Trees, and Random Forests.

OpenCV’s DNN module supports importing and running deep learning models trained in TensorFlow, PyTorch, or Caffe.

2.1.3.3 YOLO (You Only Look Once)

YOLOv10 is an advanced iteration of the YOLO object detection model, known for its speed and accuracy in real-time tasks. It not only excels in object detection but also supports image classification, semantic segmentation, and instance segmentation.

YOLOv10 utilizes a CSPNet-based backbone for improved feature extraction and faster inference. It introduces a decoupled detection head, separating classification and localization for better precision. The model uses an anchor-free approach to simplify training and improve efficiency.

YOLOv10 includes enhanced loss functions like CIoU and focal loss to address bounding box regression and class imbalance. It supports multiple image sizes, making it suitable for edge devices with limited computational power.

Unlike earlier YOLO versions, YOLOv10 is implemented in PyTorch, ensuring easier integration into modern pipelines and compatibility with hardware acceleration tools like ONNX and TensorRT.

Its high performance and accuracy make it ideal for real-time applications, including video-based tasks. YOLOv10 detects objects, draws bounding boxes, and outputs class labels and confidence scores, with fine-tuning available for specific datasets.

Model Variant	Input Size	Parameters	mAP@50	FPS (GPU)
YOLOv10n	640x640	~4M	50.1	120+
YOLOv10s	640x640	~11M	56.8	100+
YOLOv10m	640x640	~25M	63.2	75+
YOLOv10l	640x640	~40M	67.3	50+

Table 2: Performance Benchmarks of the basis YOLO models

2.1.3.4 Javascript

JavaScript is a high-level, interpreted programming language primarily known for enabling dynamic behavior in web applications. It is a core technology of the World Wide Web, alongside HTML and CSS, which are also used.

JavaScript was originally developed for client-side scripting but has evolved to support full-stack development, thanks to environments like Node.js.

JavaScript supports multiple programming paradigms, including object-oriented, imperative, and functional programming.

JavaScript runs directly in web browsers, allowing developers to build interactive user interfaces, handle events, manipulate the DOM, and communicate with servers via asynchronous requests (AJAX, Fetch API).

With the rise of frameworks and libraries like React, Angular, and Vue, JavaScript has become central to modern front-end development.

On the back end, JavaScript (via Node.js) supports building scalable network applications, REST APIs, and real-time systems using libraries like Express.js and Socket.io.

JavaScript is event-driven and non-blocking by nature, using asynchronous programming models such as callbacks, promises, and async/await to manage concurrency.

JavaScript has a vast ecosystem of packages accessible via npm (Node Package Manager), empowering developers to rapidly prototype and deploy applications.

2.1.3.5 HTML & CSS

HTML is the standard markup language for creating and structuring web content. It defines the layout and structure of web pages using nested tags, organizing elements like headings, paragraphs, images, links, lists, tables, and forms.

HTML uses a tree-like DOM (Document Object Model) structure, allowing dynamic manipulation with JavaScript and styling with CSS. HTML5 introduces features like audio/video embedding, canvas for 2D graphics, and semantic elements such as <article>, <section>, and <nav>.

HTML is platform-independent and supported by all modern browsers without external plugins, often combined with CSS and JavaScript to create fully functional websites.

CSS controls the presentation and layout of HTML elements, enabling the separation of content and design. It supports responsive design with media queries, adapting layouts to different screen sizes.

CSS applies styles like colors, fonts, margins, spacing, and positioning to create visually appealing interfaces. It is organized in a cascading manner, allowing rules to override based on specificity and order.

CSS3 adds powerful features like animations, transitions, gradients, flexbox, and grid layouts.

2.1.3.6 Google Colab

Google Colab is a cloud-based platform that helps overcome the incompatibility issues with local machines, as NVIDIA's CUDA and cuDNN services don’t work locally.

It allows users to write and execute Python code in a Jupyter notebook environment, making it easy to run machine learning, data analysis, and other Python tasks without local setup.

Key features of Google Colab include free access to GPUs and TPUs for accelerated computing, integration with Google Drive for easy access to notebooks, real-time collaboration on the same notebook, and pre-installed popular Python libraries like TensorFlow, PyTorch, NumPy, and Pandas for quick experimentation.

2.1.3.7 Roboflow

Roboflow is a web-based platform that simplifies managing, labeling, augmenting, and deploying computer vision datasets and models.

It supports tasks like object detection, classification, and segmentation, offering tools to annotate images, convert formats, and export datasets for various frameworks (e.g., YOLO, COCO, TensorFlow).

Roboflow also provides hosted training and model deployment options, allowing users to create and test models without local setup. It integrates with popular frameworks and tools, streamlining the computer vision pipeline from dataset creation to inference.

2.2 Methods

The project follows a deep learning pipeline for weapon detection:

2.2.1 Dataset Preparation

The dataset has been found from open source library named Roboflow and it likely contains images of weapons (e.g., guns, knives).

The dataset is structured in YOLO format:

```
Weapons_Dataset_new/
├── images/
│  ├── train/ (Training images)
│  ├── val/ (Validation images)
│  ├── test/ (Testing images)
├── labels/ (Bounding box labels)
├── dataset.yaml (Configuration file for YOLO training)
```

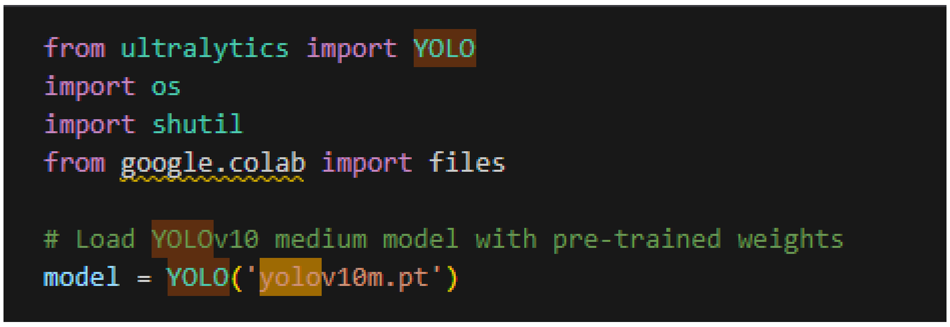
```
dataset.yaml defines like:

train: /content/Weapons_Dataset_new/images/train
val: /content/Weapons_Dataset_new/images/val
test: /content/Weapons_Dataset_new/images/test
nc: 2 # Number of classes (e.g., gun, knife)
names: ["gun", "knife"]
```

2.2.2 Installing Dependencies

Before running the YOLOv10 model, the required libraries has to be installed:

Ultralytics YOLO → Pre-trained object detection model.
OpenCV → Reads and processes images.

2.2.3 Loading & Training the YOLO Model



```python
from ultralytics import YOLO
import os
import shutil
from google.colab import files

# Load YOLOv10 medium model with pre-trained weights
model = YOLO('yolov10m.pt')
```

Image 1: Model loading

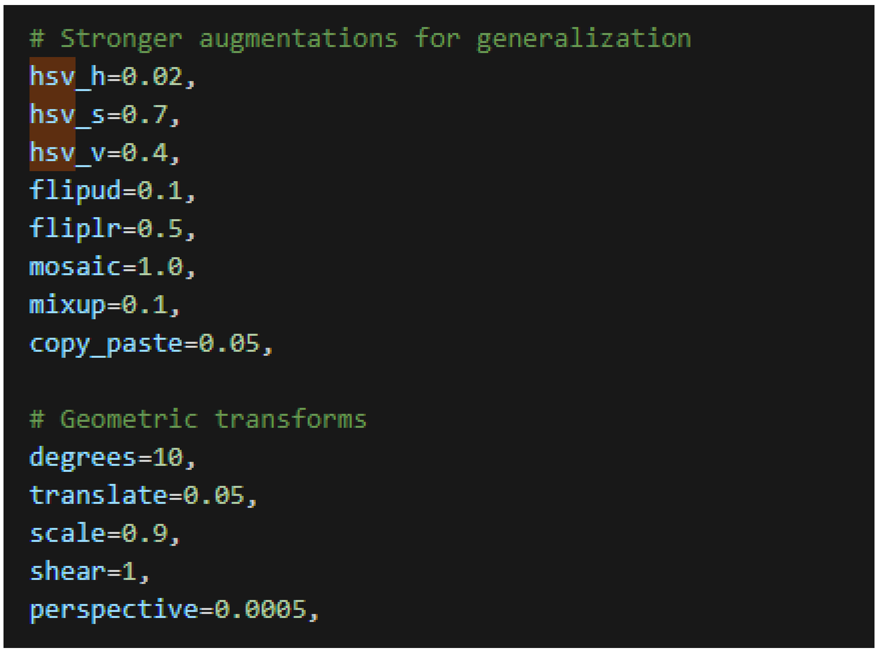
YOLOv10 model must be loaded. YOLOv10m is used because it balances speed & accuracy.

2.2.4 Augmenting Data

Augmentation types below were used.

Augmentation Type	Parameter Value
Hue, Saturation, Value	hsv_h=0.02, hsv_s=0.7, hsv_v=0.4
Flipping	flipud=0.1, fliplr=0.5
Mosaic	1.0
MixUp & CopyPaste	mixup=0.1, copy_paste=0.05
Rotation	degrees=10
Translation	translate=0.05
Scaling	scale=0.9
Shearing	shear=1
Perspective Distortion	perspective=0.0005

Table 3: Used augmentation types



```
# Stronger augmentations for generalization
hsv_h=0.02,
hsv_s=0.7,
hsv_v=0.4,
flipud=0.1,
fliplr=0.5,
mosaic=1.0,
mixup=0.1,
copy_paste=0.05,

# Geometric transforms
degrees=10,
translate=0.05,
scale=0.9,
shear=1,
perspective=0.0005,
```

Image 2: Augmentation types

These augmentations allowed the model to adapt to various lighting, angle, and background conditions without overfitting.

2.2.5 Model Training

The training process was performed using the YOLOv10m architecture, which consists of 136 layers, 15.3 million parameters and 58.9 GFLOPs computational load.

Parameter	**Value**
Epochs	250
Batch Size	8
Image Size	1280x1280
Optimizer	AdamW
Learning Rate	0.0008 (lr0)
Patience	50 (early stop)
Loss Weights	box=8.0, cls=0.4, dfl=1.5
Save Period	every 10 epochs
Validation	Enabled after each epoch

Table 4: Training Configurations

2.2.6 Model Evaluation

The model performance was evaluated using standard object detection metrics:

- Precision: True Positive / (True Positive + False Positive)
- Recall: True Positive / (True Positive + False Negative)
- mAP@0.5: Area under the precision-recall curve at 0.5 IoU
- mAP@0.5:0.95: Averaged across 0.5–0.95 IoU thresholds

Evaluation was carried out on the test set with 24 batches, showing high accuracy on the "knife" class.

2.2.7 Deployment: Web Interface

A lightweight web interface was developed using HTML, CSS, and JavaScript to allow testing of the trained model via drag-and-drop or webcam-based inputs. This UI enables:

- Image-based inference
- Video stream inference
- Upload of .jpg or .mp4 files for evaluation
- The best.pt model file was converted into a format compatible with web-based inference using JavaScript-based wrappers and served in a browser for demonstration purposes.

3. USER INTERFACE

To make the weapon detection model accessible to both technical and non-technical users, a web-based user interface (UI) was designed. This interface allows users to interact with the detection system through a browser, eliminating the need for command-line execution or local dependencies. The backend uses Flask, while the frontend is built with HTML5, CSS3, and embedded JavaScript for a responsive, interactive experience.

3.1 System Architecture Overview

The interface architecture uses a client-server model. The client-side (browser) handles user input like image upload, login, and navigation, while the server-side (Flask backend) manages model inference, authentication, routing, and responses. This modular design separates visual components from logic layers, simplifying system maintenance and expansion.

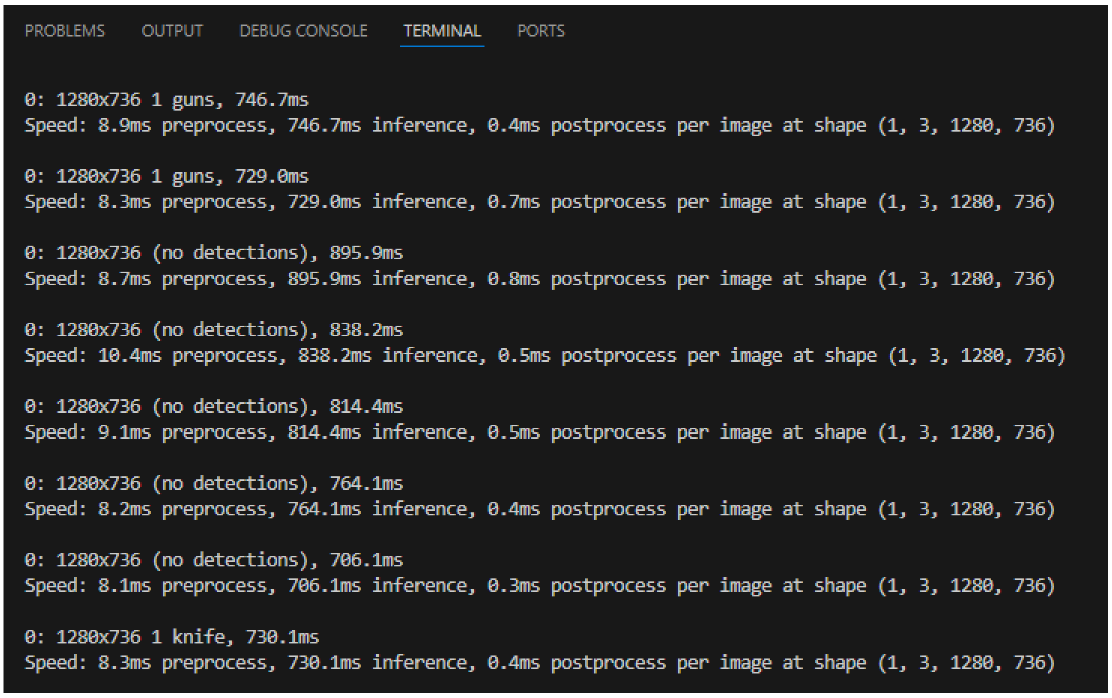
3.1.1 Backend: Flask Framework

The backend is developed in Python using the Flask microframework, which provides the following core functionalities:

- Routing and Templating: URL endpoint mapping (@app.route) to corresponding HTML templates via render_template().
- YOLOv10m Model Integration: The trained best.pt model is loaded during server startup and used for inference when users upload images.
- Image Processing Pipeline:

1. Receive uploaded image via POST request.
2. Pass it to the YOLO model.
3. Draw bounding boxes and annotate results.
4. Return prediction image to frontend.

- User Authentication System: Built-in login, signup, session control, and secure password reset.
- File Handling: Secure file upload and temporary storage using Flask's request.files and Werkzeug utilities.



```
0: 1280x736 1 guns, 746.7ms
Speed: 8.9ms preprocess, 746.7ms inference, 0.4ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 1 guns, 729.0ms
Speed: 8.3ms preprocess, 729.0ms inference, 0.7ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 (no detections), 895.9ms
Speed: 8.7ms preprocess, 895.9ms inference, 0.8ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 (no detections), 838.2ms
Speed: 10.4ms preprocess, 838.2ms inference, 0.5ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 (no detections), 814.4ms
Speed: 9.1ms preprocess, 814.4ms inference, 0.5ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 (no detections), 764.1ms
Speed: 8.2ms preprocess, 764.1ms inference, 0.4ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 (no detections), 706.1ms
Speed: 8.1ms preprocess, 706.1ms inference, 0.3ms postprocess per image at shape (1, 3, 1280, 736)

0: 1280x736 1 knife, 730.1ms
Speed: 8.3ms preprocess, 730.1ms inference, 0.4ms postprocess per image at shape (1, 3, 1280, 736)
```

Image 3: Sample Terminal Output while Detection

The app.py script serves as the central orchestrator, managing model interactions, session state, and page rendering.

3.1.2 Frontend: HTML Templates and CSS Styling

The frontend of the application is composed of a series of templates located in the templates/ directory.

File Name	Functionality Description
index.html	Main interface for uploading images and visualizing detection results.
about.html	General information about the project, its goals, and background.
contact.html	Contact form for user feedback or inquiries.
login.html	Secure login page for registered users.
signup.html	Registration page for new users.
reset_request.html	Interface for requesting a password reset email/token.
reset_token.html	Form to reset password using the provided token.

Table 5: Frontend HTML Files

The templates utilize Bootstrap-style classes and custom CSS for layout, responsiveness, and form styling. User feedback mechanisms are implemented using JavaScript alert popups and conditional rendering of model prediction results.

3.2 Authentication and Security Features

The UI includes a basic yet functional user management and authentication system:

- Password Creation: Users can create their own accounts. Passwords are securely hashed and stored.
- Session Handling: Sessions are handled via Flask's session module.
- Authentication: Users have to authenticate themselves to the system via their emails. Real-time confirmation emails are sent to users with a link to verify.
- Login: Login state determines page access; unauthenticated users are redirected to login/signup pages.

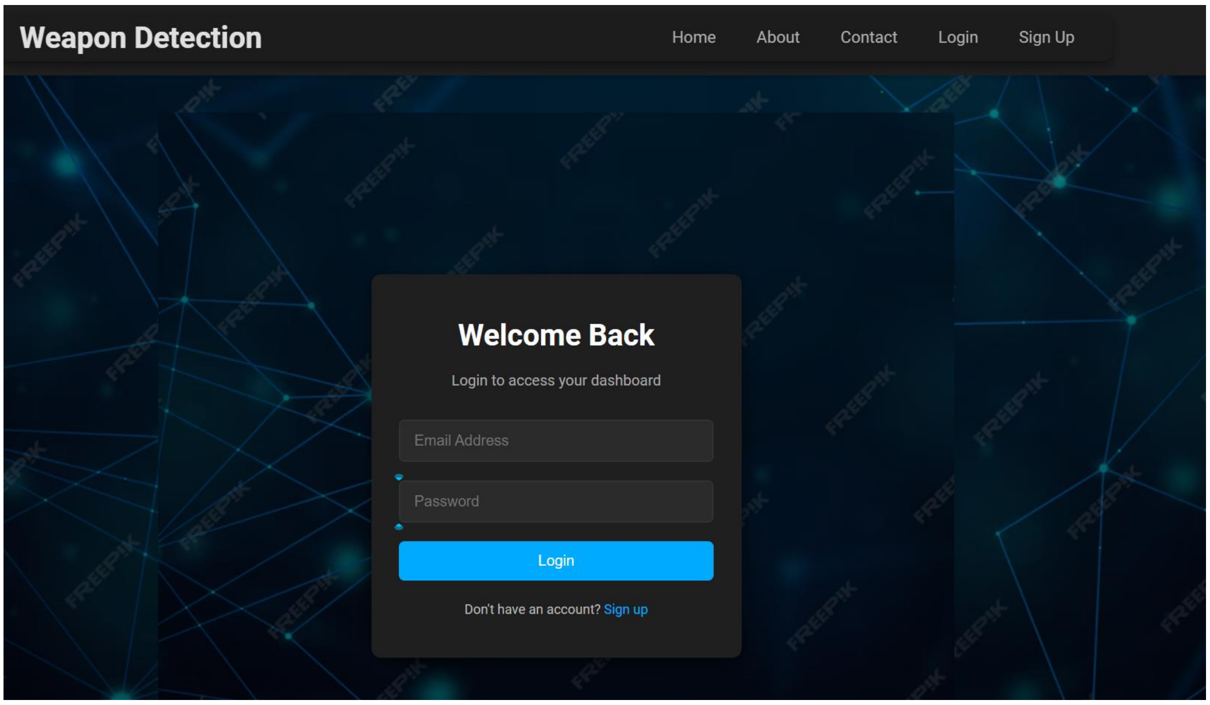
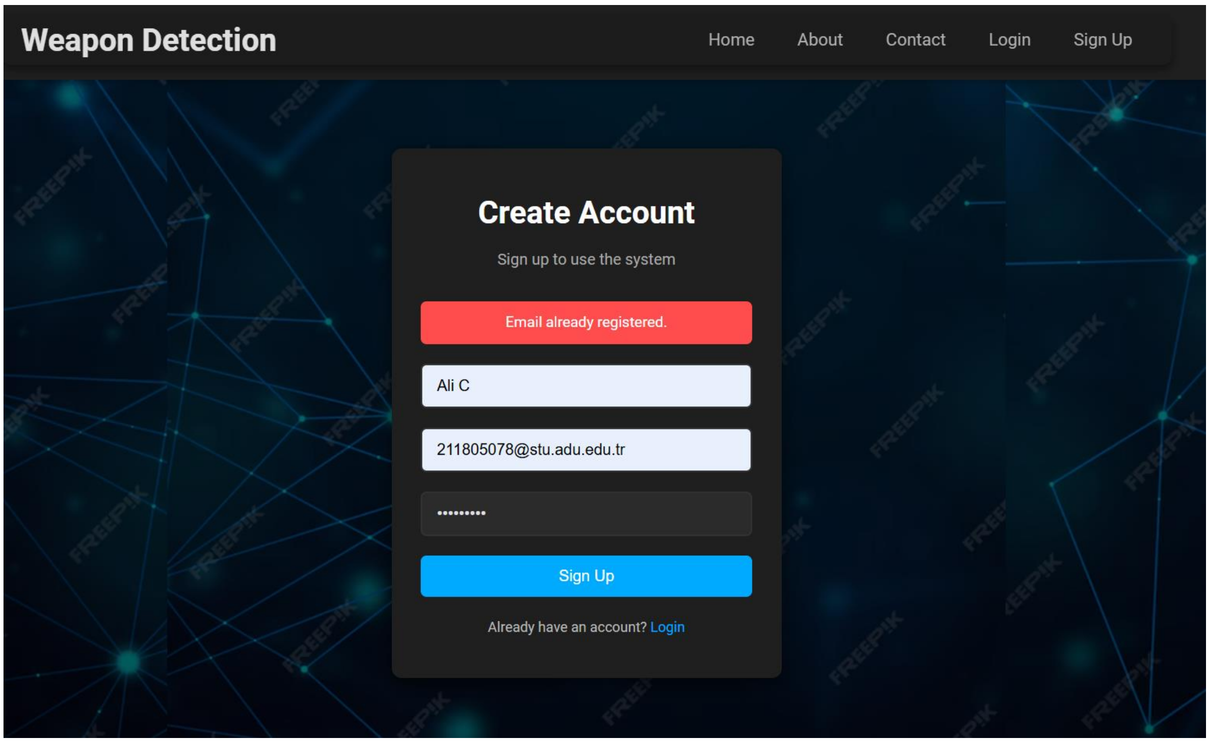


Image 4: Login page

These features ensure the system can be safely deployed in environments where access needs to be restricted (e.g., surveillance control rooms or security monitoring centers).



Email already registered.

Ali C

211805078@stu.adu.edu.tr

•••••••••

Image 5: Sign-up page

3.3 Workflow

Once authenticated, users are directed to the detection interface (index.html), where they can:

1. Upload an image or select a frame from a video.
2. Submit the media to the server.
3. View predictions in real time with bounding boxes and confidence scores.

The server-side model inference ensures privacy, as images are not shared with third parties and remain local to the server.

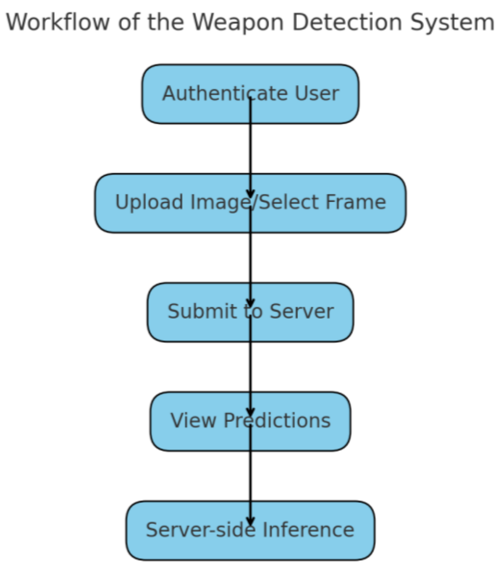


Image 6: Workflow of the Weapon Detection System

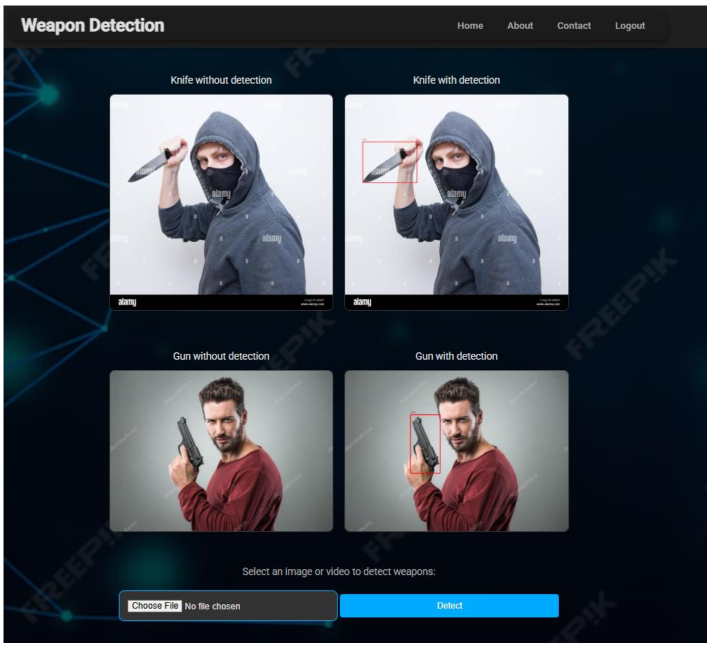


Image 7: Uploading & detection page

3.4 Future Deployment Options for User Interface

The web interface can be deployed in various ways:

- Localhost Deployment: For demonstration and testing.
- Internal Network Hosting: Suitable for use within an organization’s LAN.
- Cloud Deployment: Easily adaptable to Heroku, PythonAnywhere, or containerized via Docker.
- Integration with Surveillance Systems: The backend can be extended to process frames from RTSP-enabled CCTV feeds.

Future iterations may also include real-time video streaming and REST API endpoints for machine-to-machine integration.

3.5 Sample Use Case

A security officer at a shopping mall logs into the web interface using their registered account. After logging in, they navigate to the "Upload Image" section and submit a screenshot taken from a CCTV feed. The system processes the image immediately, detecting an object in the person’s hand within 1–2 seconds. The object is highlighted, and the system identifies it as a knife, providing a confidence score of 91%.

The officer reviews the highlighted object along with the confidence score and decides to take appropriate action. Based on the alert, they immediately notify other security personnel for further investigation. The officer then initiates the necessary security protocols to handle the potential threat. This real-time detection system enables the officer to act swiftly and make informed decisions in critical situations.

In a different scenario, if the system detects a gun or another weapon, it follows a similar process, allowing security officers to respond appropriately to various potential threats.

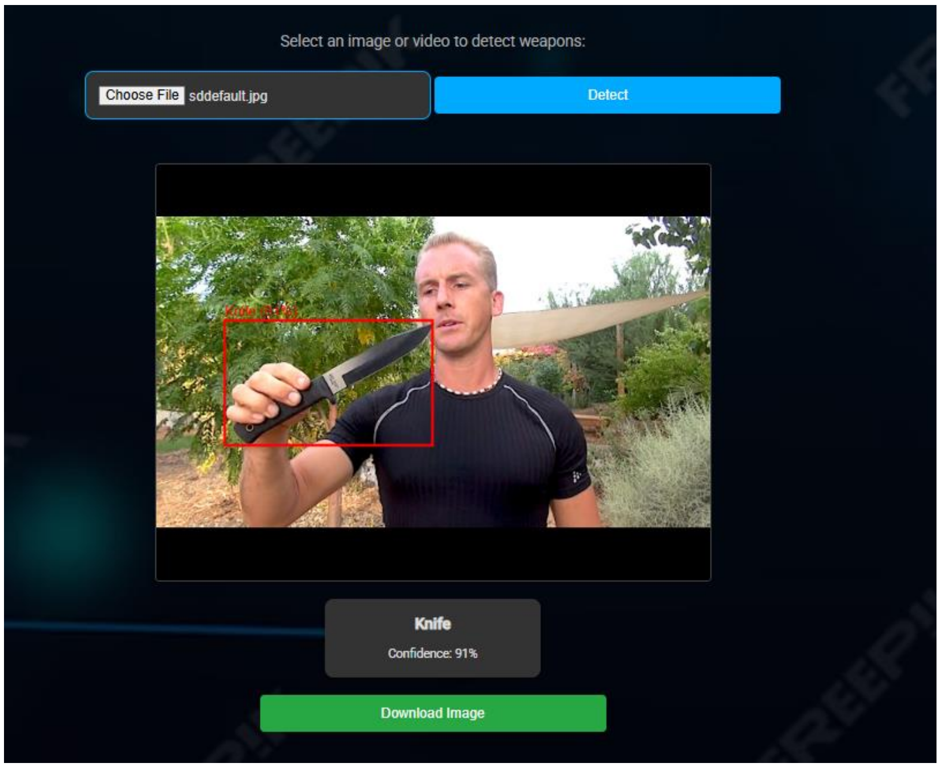

Select an image or video to detect weapons:
Choose File sddefault.jpg
Detect
Knife
Confidence: 91%
Download Image

Image 8: Sample Use Case & Detection Result

4. RESULTS & CONCLUSION

4.1 Results

The YOLOv10m model was trained for 250 epochs using the weapon dataset consisting of 383 images and 464 total weapon instances. Evaluation was conducted on the validation set, and performance metrics were obtained based on precision, recall, and Mean Average Precision (mAP) values.

4.1.1 Evaluation Results

The evaluation results are presented in the table.

Class	Images	Instances	Precision (P)	Recall (R)	mAP@0.5	mAP@0.5:0.95
All	383	464	0.892	0.877	0.911	0.583
Guns	170	193	0.863	0.819	0.856	0.425
Knives	211	271	0.920	0.935	0.965	0.741

Table 6: YOLOv10m Model Performance on Validation Dataset

4.1.2 Interpretation

The overall model performance was satisfactory, achieving 91.1% mAP@0.5, indicating high accuracy in detecting weapons. The knife class demonstrated the best performance with:

- Precision: 92.0%
- Recall: 93.5%
- mAP@0.5: 96.5%
- mAP@0.5–0.95: 74.1%

This reflects that the model was able to consistently detect knives across different backgrounds, positions, and scales with minimal false positives or negatives.

In contrast, the **gun** class yielded slightly lower metrics:

- mAP@0.5: 85.6%
- mAP@0.5–0.95: 42.5%

This disparity may be attributed to:

- Intra-class variability: Guns vary more in shape, size, and orientation compared to knives.
- Dataset imbalance: Fewer labeled images and instances for guns may have limited the model’s learning capability.
- Visual complexity: Guns are often harder to distinguish from surrounding objects in low-resolution or dark environments.

4.1.3 Real World Observations

During practical testing via the web interface:

- The system provided **instant detection (<2 seconds)** per image.
- Detection boxes were **accurate and well-aligned** with real weapon contours.
- **False positives were minimal**, especially for the knife class.
- On video-based inference (converted frames), detection consistency remained high.

These results confirm that the system is ready for real-time deployment in surveillance, checkpoint security, or public safety applications.

4.2 Conclusion

This thesis developed a real-time weapon detection system using the YOLOv10m object detection framework, aiming to detect guns and knives with high accuracy and speed through a lightweight, easily deployable architecture.

4.2.1 Key Contributions

- Integration of YOLOv10m, a state-of-the-art object detection model.
- Use of advanced augmentation techniques (mosaic, mixup, perspective) to enhance generalization.
- Training on a limited dataset (464 instances) achieving 91.1% mAP@0.5 and 58.3% mAP@0.5–0.95.
- Web interface development with Flask and HTML/CSS for real-world usability.
- End-to-end pipeline from data preparation to deployment and testing.

4.2.2 Notable Outcomes

- Knife detection outperformed gun detection, indicating room for class-specific improvements.
- The system operates in real-time with high inference speed, ideal for surveillance.
- The UI is responsive, clean, and includes user authentication for restricted access environments.

4.2.3 Future Work

Future improvements include:

- **Dataset Expansion**: Increase image diversity and balance, especially for guns and other threats (e.g., rifles, explosives).
- **Live CCTV Feed Integration**: Incorporate RTSP camera input for live video monitoring.
- **Model Compression**: Optimize for edge devices (e.g., Raspberry Pi + Coral TPU).
- **REST API Development**: Expose detection services through secure endpoints for third-party integration.

5. LITERATURE COMPARISON AND NOVELTY OF OUR WORK

In recent years, the detection of hazardous objects, particularly in the context of public safety, has been a crucial area of research where computer vision techniques are extensively applied. The YOLO (You Only Look Once) family of models is widely adopted for real-time object detection tasks. Some notable works in the literature are as follows:

5.1 Akdeniz (2023) – Object Detection with YOLOv5:

This study utilized the YOLOv5 model to explore various object detection scenarios. However, the system did not focus on real-time web interfaces or hazardous object classes (such as weapons or knives). Additionally, the training process was carried out with low-resolution and limited diversity datasets.

5.2 Roboflow Universe Weapon Detection Model (2024):

The weapon detection models published on the Roboflow platform were trained using YOLOv8 models. These models are intended for general-purpose object detection, but they lack a unique data augmentation strategy or user interface integration. Moreover, these models are mainly limited to sharing the training output without proposing a complete end-to-end system.

5.3 Ultralytics Official Documentation – YOLOv8 & YOLOv10:

The official documents from Ultralytics introduce the YOLOv10 architecture and present benchmark results. However, they mostly focus on providing example code and lack application scenarios or deployable system architectures.

5.4 Differences and Contributions of Our Work:

This thesis distinguishes itself from similar works in the literature in the following ways:

The most recent YOLOv10m architecture has been used, whereas many works in the literature still rely on YOLOv5 or YOLOv8 models.

Innovative data augmentation techniques (such as mosaic, mixup, shearing, and perspective distortion) were applied extensively, which enhanced the model's generalization capabilities.

A real-time, user-friendly web interface (built using HTML, CSS, JS + Flask backend) has been developed, a feature that is rarely found in academic studies.

A secure login system and image upload for analysis were integrated, enhancing the overall user experience.

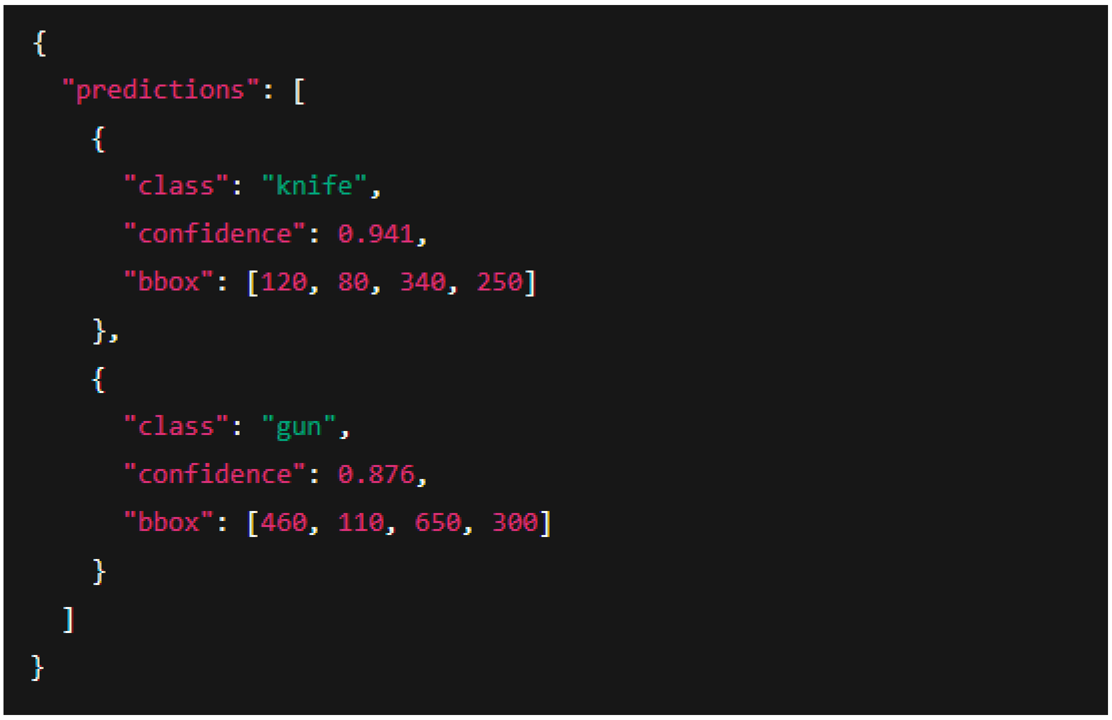
The model has demonstrated high accuracy rates, even with a small dataset: %91 mAP@0.5, and especially for the knife class, %96.5 accuracy was achieved.

6. REFERENCES

1. Glenn, J. (2023). Ultralytics YOLOv8 Documentation. Retrieved from https://docs.ultralytics.com
2. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.
3. Roboflow Universe. (2024). Weapon Detection Dataset v2. Retrieved from https://universe.roboflow.com/joao-assalim-xmovq/weapon-2
4. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.
5. OpenCV - Open Computer Vision Library
6. Google. (2024). Google Colaboratory Documentation. https://colab.research.google.com/
7. https://medium.com/@akdenizz7/yolov5-ile-nesne-tespiti-8aa370febfc0
8. https://roboflow.com/model/yolov8
9. https://roboflow.com/model/yolov10Accessed
10. https://www.youtube.com/@Roboflow

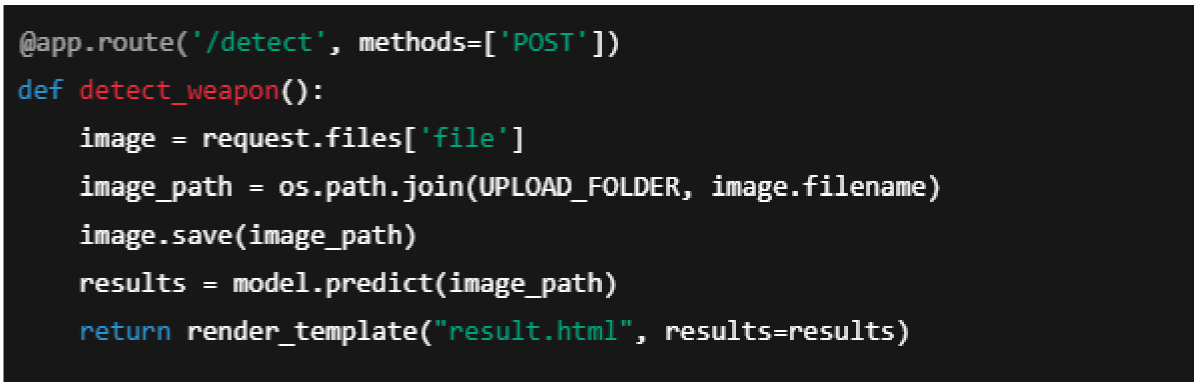
7. APPENDICES

7.1 Appendix A – Sample Detection Output JSON



```json
{
  "predictions": [
    {
      "class": "knife",
      "confidence": 0.941,
      "bbox": [120, 80, 340, 250]
    },
    {
      "class": "gun",
      "confidence": 0.876,
      "bbox": [460, 110, 650, 300]
    }
  ]
}
```

7.2 Appendix B – Sample Flask Route (Backend Inference)

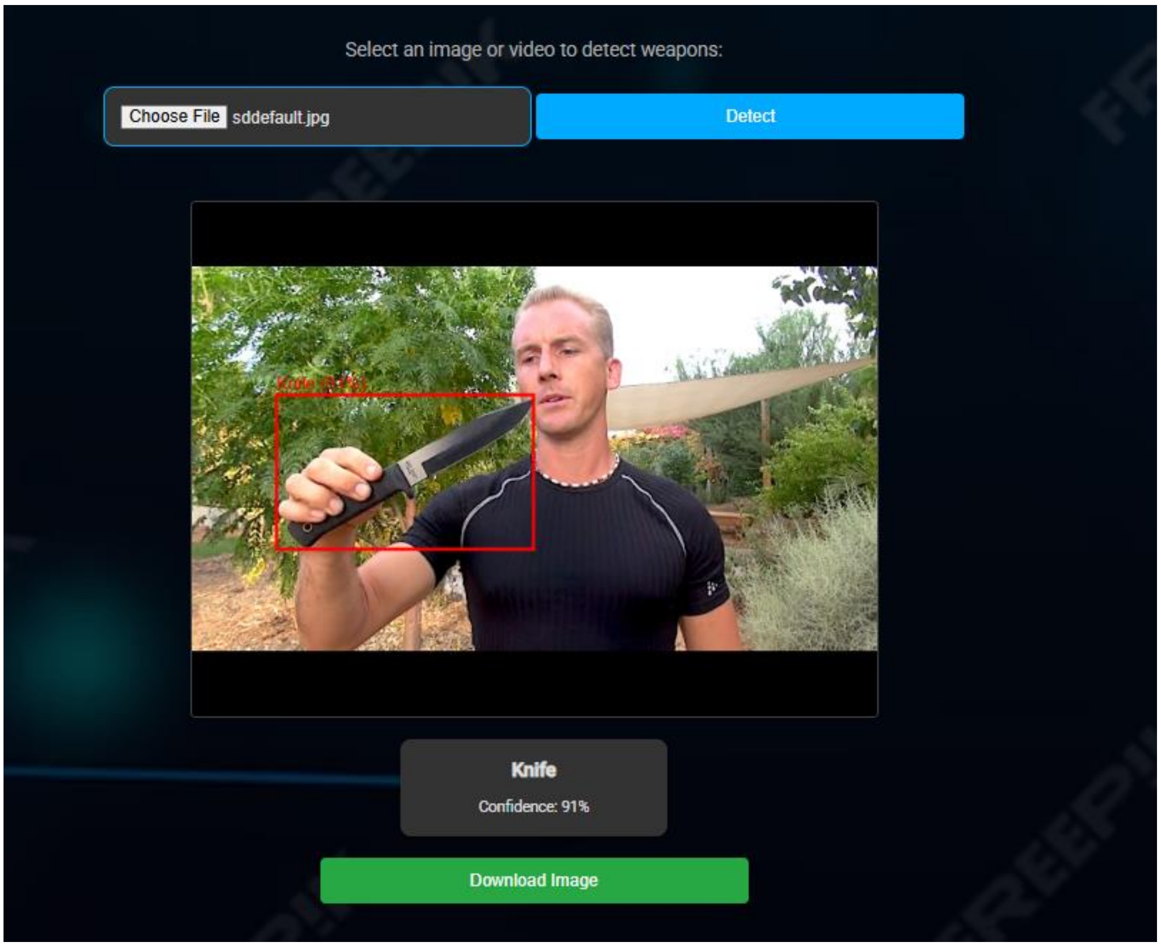


```python
@app.route('/detect', methods=['POST'])
def detect_weapon():
    image = request.files['file']
    image_path = os.path.join(UPLOAD_FOLDER, image.filename)
    image.save(image_path)
    results = model.predict(image_path)
    return render_template("result.html", results=results)
```

7.3 Appendix C – YOLO Data YAML Configuration

```yaml
train: ./images/train
val: ./images/val
test: ./images/test

nc: 2
names: ['gun', 'knife']
```

7.4 Appendix D – Web Interface Screenshot


Select an image or video to detect weapons:
Choose File sddefault.jpg
Detect
Knife (91%)
Knife
Confidence: 91%
Download Image