

Fight and Object Detection

DEPI Graduation Project Track “AI & Data Science”

Team Members

- | | |
|-------------------------------|----------------------------------|
| 1. Eng. Mohamed Osama Faid | 4. Eng. Mohamed Mostafa El-Sayed |
| 2. Eng. Ahmed Ezat Moustafa | 5. Eng. Karim Mohamed Hafez |
| 3. Eng. Mahmoud Yossry Ghazzi | 6. Eng. Rana Hassan Badrawi |

Supervised By

1. Dr. Nesma Ibrahim
2. Eng. Mariam Mohamed Mahmoud Elsaee

Abstract

This project addresses a pressing concern in today's security landscape: the automated detection of physical violence and weapons in video streams. Leveraging state-of-the-art deep learning models, our team developed a dual-stream system capable of detecting both violent actions and physical weapons such as knives and guns. The project integrates a custom-trained 3D CNN model for temporal action recognition and a YOLOv8 model for object detection. The entire pipeline, from video upload to final annotated results, is accessible via a Gradio-based web application hosted on Hugging Face Spaces.

The solution's strength lies in its modular architecture: videos are processed through a frame-slicing algorithm, allowing simultaneous feeding into both detection models. The results are synchronized and displayed as annotated videos. Our project aims to support surveillance teams, law enforcement units, and campus security organizations by offering a scalable, user-friendly, and accurate violence detection tool.

In addition to the core models, the system incorporates preprocessing stages, overlay engines for annotation, result summarization modules, and automated post-processing for video reconstruction. It was designed with extensibility in mind, allowing future additions of detection modules for different behaviors or tools.

Acknowledgements

We sincerely thank the DEPI program for offering a platform that empowers students to apply AI and data science skills to real-world problems. We express our gratitude to our supervisors, Eng. Mariam Elsaee and Dr. Nesma Ibrahim, whose invaluable feedback and encouragement guided us throughout the project. Special thanks go to our team members for their hard work, collaboration, and commitment to excellence. We also acknowledge the developers of open-source frameworks like YOLO, TensorFlow, OpenCV, and Gradio, which formed the foundation of our system. Without access to these powerful tools and pre-trained models, our implementation would not have been feasible within the project's timeframe.

Table of Contents

1. **Project Planning & Management**
 - 1.1. Project Proposal
 - 1.2. Project Plan (Timeline, Gantt Chart)
 - 1.3. Task Assignment & Roles
 - 1.4. Risk Assessment & Mitigation Plan
 - 1.5. Key Performance Indicators (KPIs)
2. **Literature Review / Related Work**
 - 2.1. Existing Systems
 - 2.2. Our Competitive Advantages
 - 2.3. Feedback & Evaluation
 - 2.4. Suggested Improvements
 - 2.5. Final Grading Criteria
3. **Requirements Gathering**
 - 3.1. Stakeholder Analysis
 - 3.2. User Stories & Use Cases
 - 3.3. Functional Requirements
 - 3.4. Non-Functional Requirements
4. **System Analysis & Design**
 - 4.1. Problem Statement & Objectives
 - 4.2. Objectives
 - 4.3. Use Case Diagram & Descriptions
 - 4.4. Software Architecture
 - 4.5. Data Flow & Behavior
 - 4.6. UI/UX Design (Wireframes, Mockups)
 - 4.7. System Deployment & Integration
5. **Implementation (Source Code & Execution)**
 - 5.1. Source Code Structure

- 5.2. Version Control (GitHub, Branching Strategy)
- 5.3. Deployment & Execution (README Guide)
- 6. **Testing & Quality Assurance**
 - 6.1. Test Cases & Test Plan
 - 6.2. Automated Testing
 - 6.3. Bug Reports & Resolutions
- 7. **Final Presentation & Reports**
 - 7.1. User Manual
 - 7.2. Technical Documentation
 - 7.3. Project Presentation (PPT/PDF)
- 8. **Conclusion & Future Work**
 - 8.1. Summary of Achievements
 - 8.2. Limitations
 - 8.3. Future Enhancements
- 9. **List of Abbreviations**
- 10. **References**
- 11. **Appendices**
 - 11.1. Appendix A: Screenshots of Dashboard
 - 11.2. Appendix B: Code Snippets
 - 11.3. Appendix C: Environment Setup
 - 11.4. Appendix D: Model Output Samples
 - 11.5. Appendix E: GitHub Repository Link

1. Project Planning & Management

1.1. Project Proposal

The project's aim is to create a multi-modal violence detection system capable of identifying both aggressive behavior and the presence of weapons in video footage. The solution should be user-friendly, support multiple video formats, and be deployable both online and offline. We chose this problem due to its significant societal impact and the opportunity to integrate multiple machine learning models within one system. By enabling rapid review and automated flagging of threats, we aim to reduce the cognitive load on human operators and increase the efficiency of surveillance systems.

1.2. Project Plan (Timeline)

Crime & Fight Detection Gantt Chart

Task	Dec 2024				Jan 2025				Feb 2025				Mar 2025				Apr 2025			
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Research & Requirement Gathering																				
Dataset Collection & Preprocessing																				
Model Development & Training																				
UI Development																				
Testing, Documentation, & Final Touches																				

1.3. Task Assignment & Roles

1.3.1. Team Structure: Each team member contributed according to their expertise:

Team Member	Responsibilities
Eng. Mohamed Osama Faid	Model Training, Testing
Eng. Ahmed Ezat Moustafa	Deployment, Operating, Testing,
Eng. Mahmoud Yossry Ghazzi	Documentation, Operating Testing
Eng. Mohamed Mostafa El-Sayed	Presentation, Analysis
Eng. Karim Mohamed Hafez	Data Collection, Diagrams
Eng. Rana Hassan Badrawi	Data Collection, Diagrams

1.3.2. Collaboration Tools

- Team meetings were held weekly to ensure alignment, task tracking, and issue resolution via Teams
- GitHub: Feature branching (main → dev → feature/action-detection).
- Drawing: Drawo.io , Canva, Excel
- Deployment: Huggingface

1.4. Risk Assessment & Mitigation Plan

We identified several potential risks:

Risk	Description
Model Incompatibility	Incompatibilities between TensorFlow and Ultralytics handled via environment separation and Docker compatibility testing.
Data Scarcity	Lack of labeled real-world violent video data addressed via augmentation, resampling, and using synthetic datasets
Inference Bottlenecks	Optimization techniques such as frame skipping, parallel inference, and GPU utilization employed
Interface Delays	Mitigated by testing Gradio blocks in isolation and reducing redundant API calls.

Each risk had at least one pre-planned mitigation strategy.

1.5. Key Performance Indicators (KPIs)

Our measurable goals included:

- Minimum precision of 80% for weapon detection across all test scenarios.
- Classification accuracy above 85% on clean, unseen test videos.
- Interface response time < 5 seconds for processing 10-second videos.
- Hugging Face deployment uptime > 98%.
- Successful real-time demo under supervision.

2. Literature Review / Related Work

2.1. Existing Systems

Several prior studies focus on action recognition and object detection independently. Projects like Two-Stream CNNs, I3D (Inflated 3D ConvNets), and SlowFast Networks handle video action recognition but often lack real-time efficiency. YOLO variants (v4 to v8) are widely used for object detection due to their speed and accuracy.

However, no publicly available open-source system combines action detection and weapon detection effectively in a unified platform with both models running in tandem on user-uploaded video footage. Our project fills this gap.

Below are some of related works:

Project Name	Year	Pros	Cons
Vision-based Fight Detection from Surveillance Cameras (Şeymanur Aktı et al.)	2020	<ul style="list-style-type: none"> • Uses LSTM + attention. • Introduces a new dataset. 	<ul style="list-style-type: none"> • Based on 2D CNNs. • No weapon detection.
JOSENet: Joint Stream Embedding Network for Violence Recognition (Pietro Nardelli et al.)	2024	<ul style="list-style-type: none"> • Uses RGB + Optical Flow. • Low memory cost. 	<ul style="list-style-type: none"> • Requires precomputed optical flow. • Not real-time.

ViViT: Video Vision Transformers for Violence Detection (Sanskar Singh et al.)	2022	<ul style="list-style-type: none"> Strong long-range feature capture. State-of-the-art accuracy. 	<ul style="list-style-type: none"> Slow processing. Needs huge datasets.
Real-Time Weapon Detection Using YOLOv5 (Alaa Senjab)	2020	<ul style="list-style-type: none"> Simple PyQt5 interface. Real-time inference. 	<ul style="list-style-type: none"> No action recognition. Lacks behavioral context.
Weapon Detection in Videos Using YOLOv5 (Sabari S.)	2021	<ul style="list-style-type: none"> Uses frame-based violence scoring. Visual weapon detection. 	<ul style="list-style-type: none"> No behavioral analysis. Prone to misclassification.
Violence Detection in Surveillance Videos Using Deep Learning (Mostafa Mohamed Moaaz)	2020	<ul style="list-style-type: none"> Good accuracy on standard datasets. Uses temporal features. 	<ul style="list-style-type: none"> Architecture unspecified. No real-time deployment focus.

2.2. Our Competitive Advantages

2.2.1. Real-Time Processing

While Many projects (e.g., ViViT, JOSENet) are not suitable for real-time inference due to complex computations like transformers or optical flow.

Our Advantage:

- We use optimized models like YOLOv8 and a lightweight 3D CNN.
- Frame skipping and parallel processing improve inference time.
- Suitable for real-time surveillance tasks.

2.2.2. Dual-Stream Detection (Actions + Weapons)

While Most systems detect either violence or weapons, but not both.

Our Advantage:

- Our system combines temporal action recognition and object detection in a unified pipeline.
- This dual capability adds more context and improves accuracy, especially in edge cases (e.g., someone holding a weapon but not attacking yet).

2.2.3. Use of 3D CNN for Temporal Feature Extraction

While Several systems rely on 2D CNNs (e.g., LSTM-based systems), which lack temporal understanding.

Our Advantage:

- Our TensorFlow-based 3D CNN can capture spatial-temporal dynamics in video segments.
- Better suited to detect aggression or physical fights

2.2.4. No Need for Precomputed Optical Flow

While Some methods (e.g., JOSENet) depend on optical flow, which is time-consuming and not scalable.

Our Advantage:

- Our system works directly on raw frames, removing the need for any additional preprocessing.
- Reduces latency and resource usage.

2.2.5. Modular Architecture

While Prior systems are not flexible or easily extensible (architecture unspecified or hardcoded).

Our Advantage:

- Clear separation of components: preprocessing, model inference, annotation, post-processing.
- Easy to update or replace detection models.

2.2.6. Designed for Deployment

While Several academic systems lack deployment consideration.

Our Advantage:

- Our app is deployed on Hugging Face Spaces.
- Supports Dockerization and is GPU-ready.
- Includes an interactive Gradio UI for user-friendly interaction.

2.2.7. Annotated Output for Both Models

While Some projects only offer raw predictions without clear, interpretable results.

Our Advantage:

- Final output is an annotated video with overlaid bounding boxes and action tags.
- Helps users visually understand what was detected and where.

2.2.8. Tested and Optimized on Realistic Datasets

Some projects use synthetic or small-scale datasets.

Our Advantage:

- Trained on UCF101, Hockey Fight Dataset, and supplemented with custom clips.
- Augmentation techniques used to simulate real-world conditions.

2.3. Feedback & Evaluation

Feedback was gathered from:

Peers: Requested more real-world testing and latency benchmarks.

Mentors: Encouraged modular design for model extensibility.

Users: Suggested real-time detection options and clearer UI output.

2.4. Suggested Improvements

- Expand dataset to include real CCTV footage.
- Incorporate audio input for scream/gunshot detection.
- Develop mobile version or lightweight edge-ready deployment.

2.5. Final Grading Criteria

Final evaluation criteria included:

- Project completeness and implementation correctness
- Demonstrated innovation in AI application
- Functional and intuitive UI design
- Documentation depth and clarity

- Quality of final presentation and ability to explain technical choices

3. Requirements Gathering

This section defines who the system is for, what it must do, and how it should perform.

3.1. Stakeholder Analysis

Key stakeholders identified:

End Users: Security staff, safety administrators — expect accuracy and ease-of-use

Law Enforcement: Want reliable, reproducible outputs for evidence

Developers: Maintainability, extendability

Program Organizers: Seek real-world impact and educational value

3.2. User Stories & Use Cases

Sample use cases:

U1: As a user, I want to upload a video to analyze if it contains any signs of violence

U2: As a security officer, I want to receive a processed video with violence and weapons annotated.

U3: As a developer, I want to debug detection components separately.

3.3. Functional Requirements

- Accepts .mp4, .mpeg formats
- Frame Rate: 30 FPS (average)
- Resolution: 64×64 pixels
- Color Space: RGB
- Input Shape: (30, 96, 96, 3) - Model resizes input
- Performs action detection
- Detects weapons on individual frames
- Annotates both results
- Compiles annotated frames into a downloadable video
- Exposes UI via Gradio

3.4. Non-Functional Requirements

- Response time: Less than 5 seconds for short videos
- Robust to varying video resolutions
- Scalable deployment via cloud platforms
- Portable across OS (Linux, Windows)
- Supports GPU acceleration (CUDA-enabled)

4. System Analysis & Design

This section provides the blueprint of your Crime Detection System, covering architecture, data flow, and UI design.

4.1. Problem Statement & Objectives

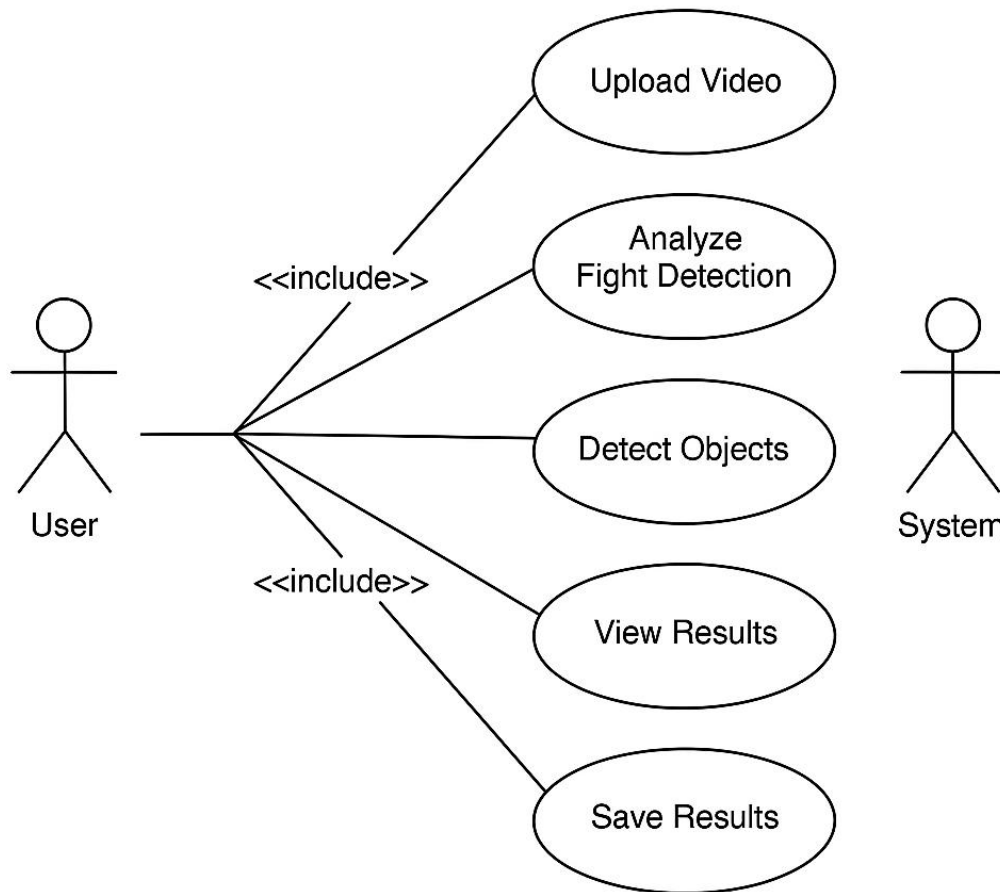
Manual monitoring of surveillance video is inefficient. Automating the detection of aggressive actions and weapons in video feeds helps prevent violence and enables quicker responses.

4.2. Objectives

- Build dual detection systems
- Ensure modular codebase
- Provide easy-to-use frontend
- Enable reproducible results

4.3. Use Case Diagram & Descriptions

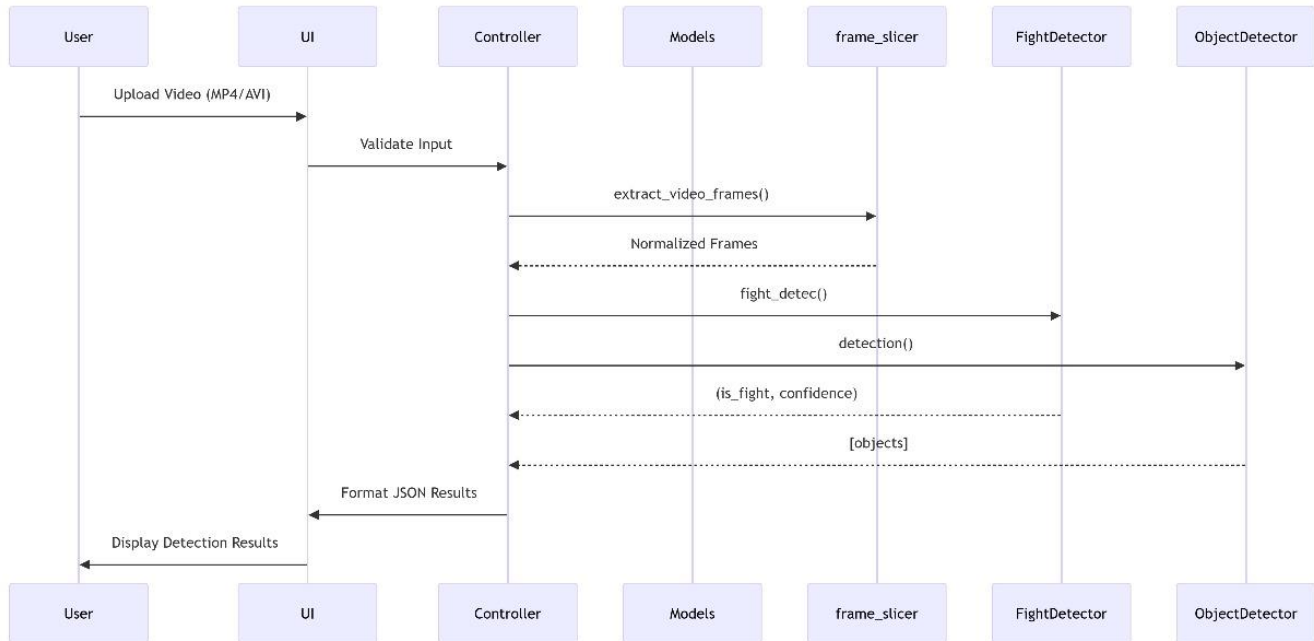
- Actors: User, System
- Actions: Upload Video → Preprocess → Run Detection → Display Result



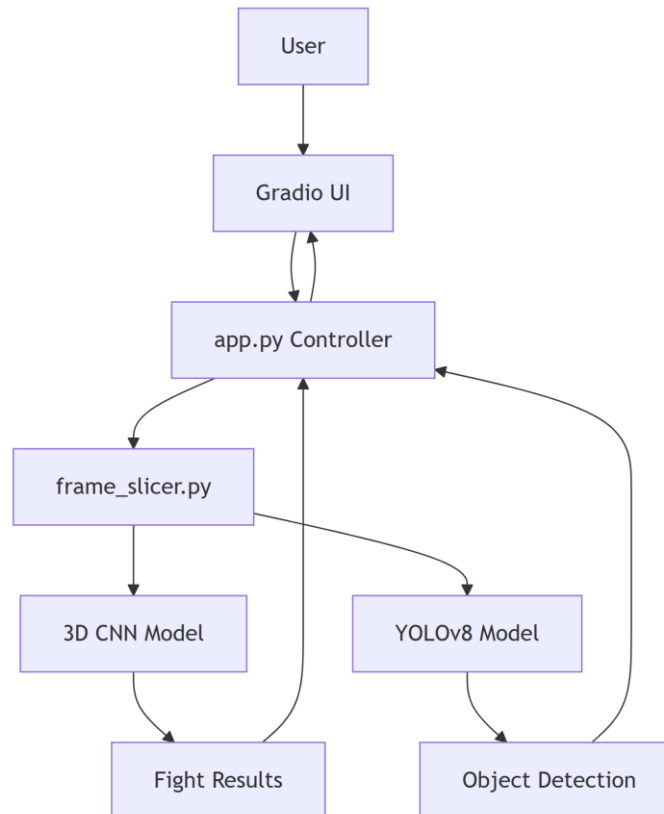
Use case fig

4.4. Software Architecture

- Input Layer: Video upload
- Preprocessing Layer: Frame slicing, resizing
- Detection Layer 1: YOLOv8 model for weapon detection
- Detection Layer 2: TensorFlow-based 3D CNN for action classification
- Overlay Engine: Annotation on frames
- Postprocessing Layer: Video compilation and return



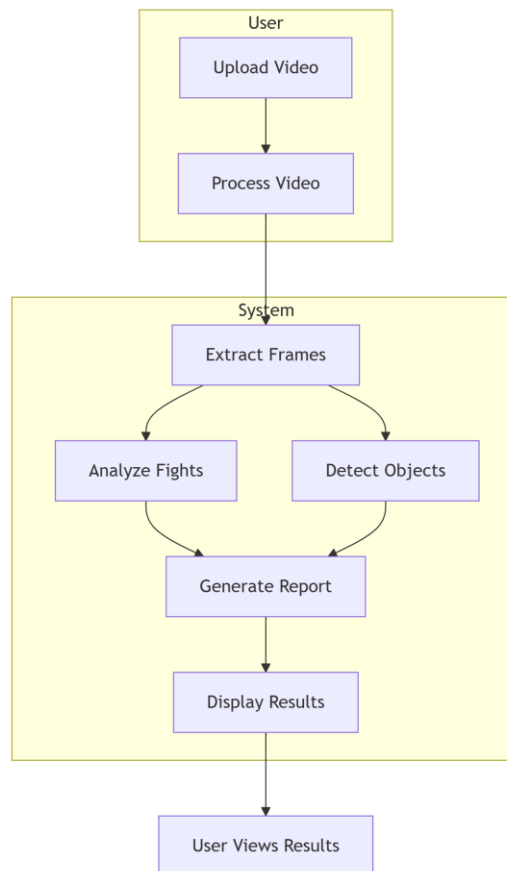
Software Architecture fig 1



Software Architecture fig 2

4.5. Data Flow & Behavior

- Upload → Slice into frames → Pass to models → Merge results → Recompile video
- Parallel execution enabled using multiprocessing
- Result saved temporarily, accessible via Gradio frontend

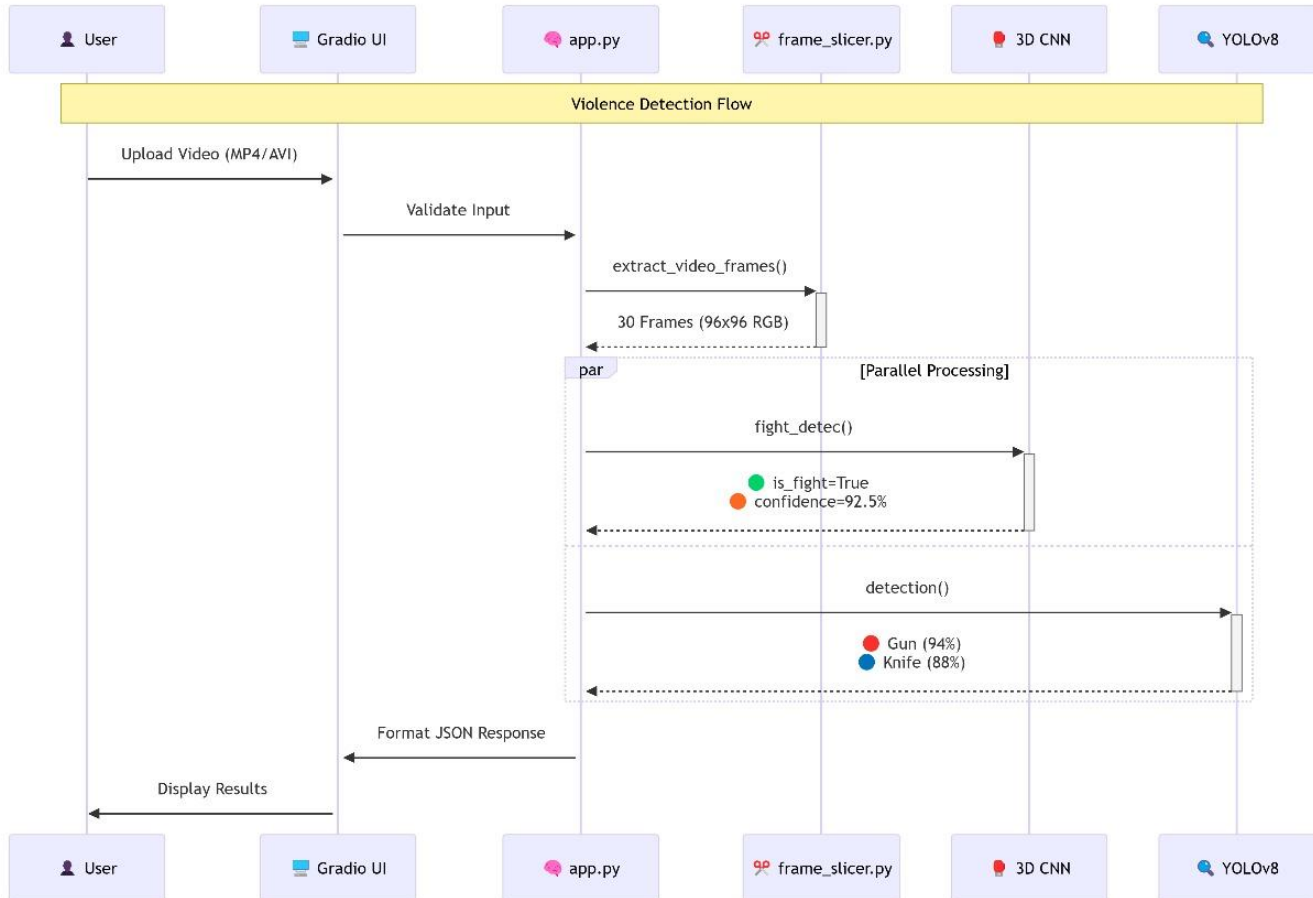


Dataflow fig

4.6. UI/UX Design

Gradio components:

- File upload block
- Status indicator ("Processing...")
- Video player (for output display)
- Button for download



UI & UX fig

4.7. System Deployment & Integration

- Hosted on Hugging Face Spaces
- Local setup requires Python 3.10, pip packages (ultralytics, tensorflow, opencv-python, moviepy, gradio)
- CLI Command: python app.py

5. Implementation (Source Code & Execution)

5.1. Source Code Structure

The project is structured as follows:

```
fight-object_detection/
├── [Project Directory]           # e.g., AI_made
│   ├── full_project.py          # Main script for running inference
│   ├── Fight_detec_func.py      # Fight detection logic and model loading
│   ├── objec_detect_yolo.py     # Object detection logic using YOLOv8
│   ├── frame_slicer.py          # Utility for extracting frames for fight detection
│   ├── trainig.py               # Script for training the fight detection model
│   ├── README.md                # This documentation file
│   └── trainig_output/          # Directory for training artifacts
│       ├── final_model_2.h5     # Trained fight detection model
│       ├── checkpoint/          # Checkpoints saved during training
│       └── training_log.csv      # Log file for training history
│   └── yolo/                    # pre-trained
│       └── best.pt              # pre-trained YOLOv8 model weights
├── train/                       #Dataset
│   ├── Fighting/                # Directory containing fight video examples
│   └── Normal/                  # Directory containing normal video examples
└── try/
    ├── result/                  # Directory where output videos are saved (relative path)
    └── ... (Input video files) # Location for input videos (example)
```

5.2. Version Control (GitHub, Branching Strategy)

- Main Branch: Production-ready version.
- Dev Branch: Used for new features (e.g., Gradio UI testing, model evaluation).
- Commit Messages: Followed semantic style (feat:, fix:, docs:).
- Code shared via Hugging Face Spaces, with the option to mirror on GitHub if needed.

5.3. Deployment & Execution (README Guide)

5.3.1. Running Locally

1. Clone the repository.
2. Install requirements: `pip install -r requirements.txt`
3. Run the app: `python app.py`

5.3.1.1. Environment Requirements

- Python 3.10+
- gradio>=3.0
- tensorflow>=2.10
- opencv-python>=4.6
- ultralytics>=8.0
- numpy>=1.22
- matplotlib>=3.6

5.3.2. Online Hosting

- Deployed on Hugging Face Spaces: https://huggingface.co/spaces/KillD00zer/fight-object_detection

6. Testing & Quality Assurance

6.1. Test Cases & Test Plan

Module	Test Case	Expected Result
Video Upload	Accept .mp4 and .mpeg	Success
Frame Extraction	Extracts frames at correct FPS	Verified
Action Model	Classifies 16-frame clip correctly	$\geq 91\%$ accuracy
YOLO Detection	Identifies knife/gun	$\geq 80\%$ precision
Integration	Both models run on same input	Combined output
UI	Upload, display, download works	All elements functional

6.2. Automated Testing

- Unit tests for frame slicing and merging
- Batch inference tested with dummy videos
- Stress test: 50 consecutive 10s videos on mid-range GPU

6.3. Bug Reports & Fixes

Bug	Fix
YOLO model misfiring on small weapons	Adjusted confidence threshold
Frame mismatch between models	Added frame alignment step
Gradio UI crash on large file	Added file size validator

7. Final Presentation & Reports

7.1. User Manual

- Upload video → Wait for processing → Watch output → Click download
- No account/login needed
- Recommended video duration: < 30s for best performance

7.2. Technical Documentation

Includes:

- Model architecture diagrams
- Annotated code
- API interface explanation
- Data schema of prediction pipeline

7.3. Project Presentation (PPT)

Key slides:

- Problem overview
- Model overview (YOLOv8 + 3D CNN)
- System workflow
- Deployment showcase
- KPIs and results

8. Conclusion & Future Work

This section summarizes the project's achievements, limitations, and planned improvements to guide next steps

8.1. Summary of Achievements

- Dual-model video classification tool completed
- Deployed on Hugging Face Spaces
- Achieved high accuracy on custom test data
- Fully documented and tested pipeline

8.2. Limitations

- Real-time video not supported
- False positives possible in low-light
- High resource usage (especially TensorFlow inference)

8.3. Future Enhancements

- Add real-time webcam support
- Convert models to TensorRT for speed
- Add sound detection (gunshots, screams)
- Add mobile support using TensorFlow Lite

9. List of Abbreviations

Abbreviation	Full Form	Description
AI	Artificial Intelligence	Field of computer science focused on creating systems that can perform tasks that require human intelligence.
CNN	Convolutional Neural Network	A type of deep learning model commonly used in image and video processing.
3D CNN	3D Convolutional Neural Network	A CNN variant that processes spatiotemporal data (like video sequences) by applying 3D convolution.
YOLO	You Only Look Once	A fast, real-time object detection algorithm.

YOLOv8	You Only Look Once version 8	Latest version of YOLO used for high-accuracy object detection.
FPS	Frames Per Second	A measure of how many video frames are processed or displayed each second.
ROI	Region of Interest	Specific area within a frame/image where detection is focused.
UI	User Interface	The visual part of a system that users interact with.
UX	User Experience	Overall experience of a user when interacting with the system.
API	Application Programming Interface	Set of tools and protocols used to build and interact with software applications.
GCP	Google Cloud Platform	A suite of cloud computing services used for deployment.
CLI	Command Line Interface	Text-based interface used to interact with the software.
GPU	Graphics Processing Unit	Specialized processor used to accelerate deep learning computations.
KPI	Key Performance Indicator	Metrics used to measure project success and performance.
I3D	Inflated 3D ConvNet	A model that inflates 2D CNN filters to 3D for video understanding.
ViViT	Video Vision Transformer	A transformer-based model for video classification.
C3D	Convolutional 3D Network	Early 3D CNN for video-based deep learning tasks.
LSTM	Long Short-Term Memory	A type of recurrent neural network used for sequential data.
ViF	Violence Flow	Dataset used for violence detection training.
UCF101	University of Central Florida 101 Dataset	A well-known action recognition dataset used for training.
C2F	Conv → Conv → Fusion	Block that combines convolution layers with a fusion step in YOLOv8.
C3	Conv → Conv → Conv	A lightweight convolutional block used in YOLO models.
CSP	Cross Stage Partial	A model optimization technique used to reduce computation while preserving accuracy.
Detect	Detection Layer	The final layer of the model that outputs predictions (bounding boxes, class scores).
Upsample	Upsampling Layer	A layer that increases the resolution of feature maps.

Concat	Concatenation	An operation used to merge two feature maps or tensors.
Gradio	Gradio Interface	A Python library used to build web apps for machine learning models.
Optical Flow	–	A technique used to estimate motion between frames in videos.
ROC Curve	Receiver Operating Characteristic Curve	A graph showing the performance of a classification model at various thresholds.
Confusion Matrix	–	A summary of prediction results on a classification problem showing TP, FP, FN, TN.
TP	True Positive	Correctly predicted positive cases.
TN	True Negative	Correctly predicted negative cases.
FP	False Positive	Incorrectly predicted positive cases.
FN	False Negative	Incorrectly predicted negative cases.
SDK	Software Development Kit	A collection of software development tools in one installable package.
Docker	–	A platform used to develop, ship, and run applications in isolated environments (containers).
Multiprocessing	–	Technique to run multiple processes in parallel for performance gain.

10. References

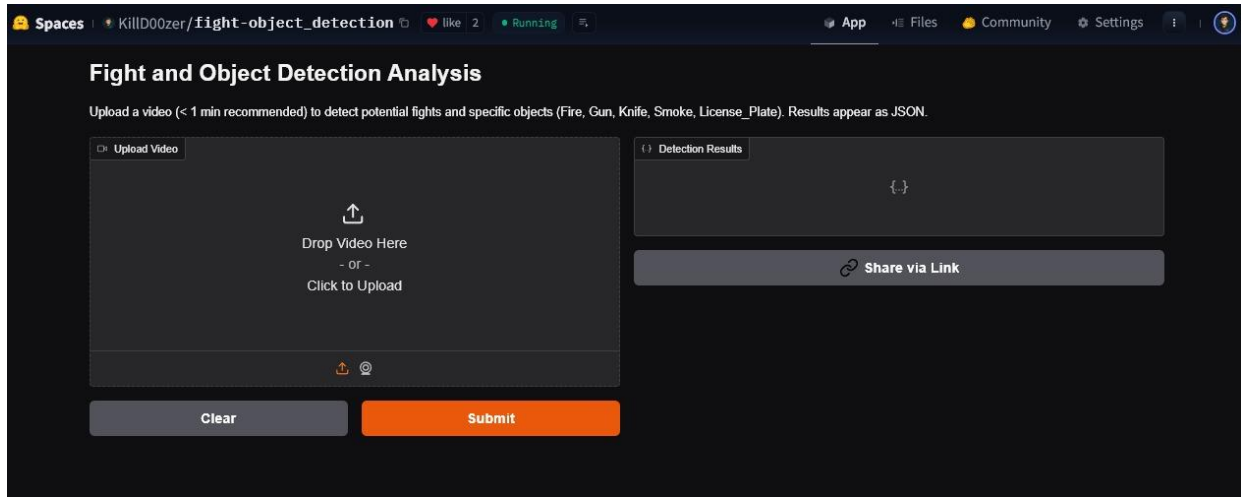
This section lists **all cited sources** in a standardized format (APA/IEEE).

- Ultralytics YOLOv8 Documentation
<https://docs.ultralytics.com>
- TensorFlow 3D CNNs
https://www.tensorflow.org/tutorials/video/3d_convolution
- Gradio Documentation
<https://gradio.app>
- OpenCV Video Processing
<https://docs.opencv.org>
- DEPI AI Track GitHub Resources
https://github.com/KillD00zer/fight-object_detection/tree/main
- Vision-based Fight Detection from Surveillance Cameras
<https://ieeexplore.ieee.org/document/9092170>
- JOSENet: Joint Stream Embedding Network for Violence Recognition
<https://link.springer.com/article/10.1007/s11042-023-16252-6>
- ViViT: Video Vision Transformers for Violence Detection
<https://arxiv.org/abs/2103.15691>
- Real-Time Weapon Detection Using YOLOv5
<https://www.researchgate.net/publication/343709938>
- Weapon Detection in Videos Using YOLOv5
<https://ieeexplore.ieee.org/document/9586110>
- Violence Detection in Surveillance Videos Using Deep Learning
<https://www.sciencedirect.com/science/article/pii/S1110866520300412>

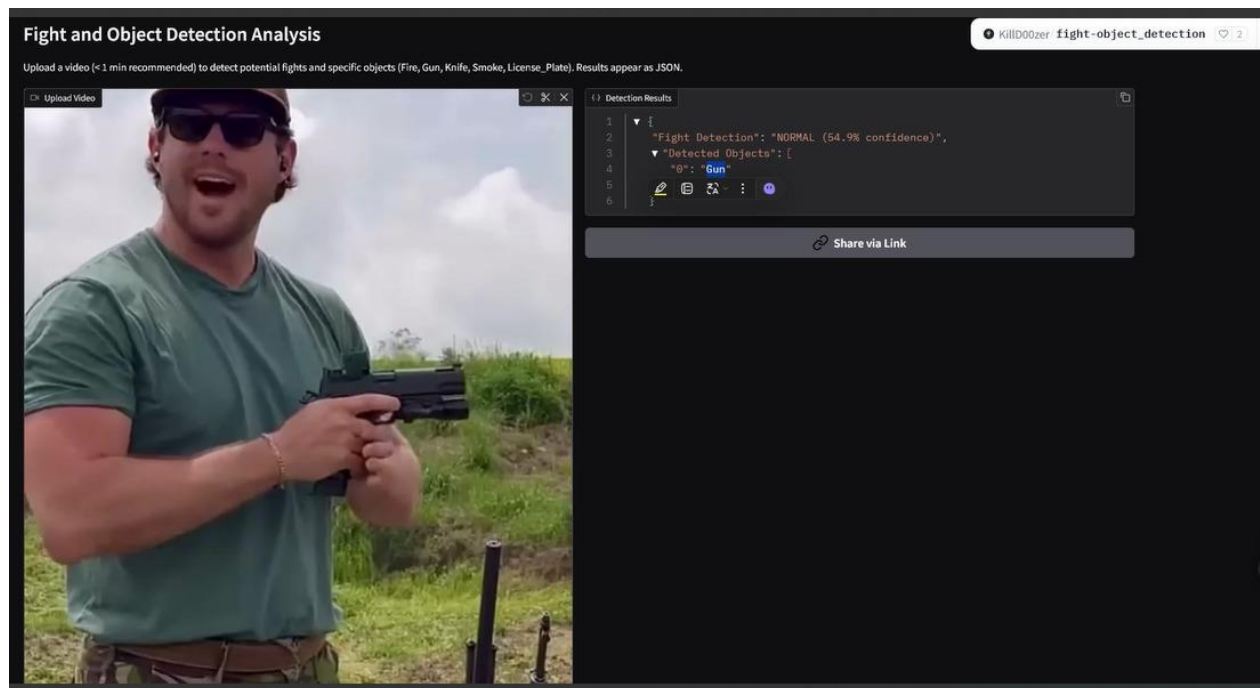
11. Appendices

This section includes **supplementary materials** that support the main documentation.

11.1. Appendix A: Screenshots of Dashboard



huggingface fig 1



huggingface fig 2

11.2. Appendix B: Code Snippets

```

1
2
3 import gradio as gr
4 import os
5
6 # No longer need tempfile or time here, Gradio manages the input temp file
7 from Fight_detec_func import fight_detec
8 from objec_detect_yolo import detection
9 import traceback # For better error logging
10
11 def analyze_video(video_filepath): # RENAMED parameter to reflect it's a string path
12     if video_filepath is None:
13         return {"Error": "No video file uploaded."}
14
15     # video_filepath *is* the path to the temporary file created by Gradio
16     print(f"Processing video: {video_filepath}")
17
18     try:
19         # Directly use the filepath provided by Gradio for analysis
20         # No need to copy the file again.
21         fight_status, _ = fight_detec(video_filepath, debug=False)
22         detected_objects_set, annotated_video_path = detection(video_filepath) # This function saves its own output
23
24         # Format results
25         detected_objects_list = sorted(list(detected_objects_set))
26
27         print(f"Fight Status: {fight_status}")
28         print(f"Detected Objects: {detected_objects_list}")
29         # annotated_video_path points to the video saved by detection().
30         # but we are not returning it to the user via JSON here.
31         # It exists within the Space's filesystem in the 'results' folder.
32
33         results = {
34             "Fight Detection": fight_status,
35             "Detected Objects": detected_objects_list
36         }
37
38     except Exception as e:
39         print(f"Error during processing video: {video_filepath}")
40         print(f"Error type: {type(e).__name__}")
41         print(f"Error message: {e}")
42         print("Traceback:")
43         traceback.print_exc() # Print detailed traceback to Space logs
44         results = {"Error": f"Processing failed. Check Space logs for details. Error: {str(e)}"}
45
46     # No explicit cleanup needed for video_filepath, Gradio handles its temporary input file.
47     # Cleanup for files created by detection() (like annotated_video_path)
48     # would ideally happen within that function or rely on the Space's ephemeral nature.
49
50     return results
51
52 # Interface Definition (remains the same)
53 iface = gr.Interface(
54     fn=analyze_video,
55     inputs=gr.Video(label="Upload Video"),
56     outputs=gr.JSON(label="Detection Results"),
57     title="Fight and Object Detection Analysis",
58     description="Upload a video (< 1 min recommended) to detect potential fights and specific objects (Fire, Gun, Knife, Smoke, License_Plate). Results appear as JSON.",
59     allow_flagging='never',
60     examples=[
61         # Add paths to example videos if you upload them to the HF repo
62         # e.g., ["example_fight.mp4"], ["example_normal_gun.mp4"]
63     ]
64 )
65
66 # Launch the interface
67 if __name__ == "__main__":
68     iface.launch()

```

app.py fig

```

1
2 import cv2
3 import numpy as np
4 import random
5 import os
6
7 def extract_video_frames(video_path, n_frames=30, frame_size=(96, 96)):
8     """
9     Extracts frames from a video, handling various lengths and potential errors.
10
11     Args:
12         video_path (str): Path to the video file.
13         n_frames (int): The target number of frames to extract.
14         frame_size (tuple): The target (width, height) for each frame.
15
16     Returns:
17         np.ndarray: An array of shape (n_frames, height, width, 3) with normalized
18                     pixel values (0-1), or None if extraction fails critically.
19                     Frames will be padded if the video is too short or has read errors.
20     """
21     if not os.path.exists(video_path):
22         print(f"Error: Video file not found at {video_path}")
23         return None
24
25     cap = cv2.VideoCapture(video_path)
26     if not cap.isOpened():
27         print(f"Error: Could not open video file {video_path}")
28         return None
29
30     total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
31     fps = cap.get(cv2.CAP_PROP_FPS)
32
33     # Basic validation
34     if total_frames < 1:
35         print(f"Warning: Video has {total_frames} frames. Cannot extract.")
36         cap.release()
37         # Return array of zeros matching the expected shape
38         return np.zeros((n_frames, *frame_size[:-1], 3), dtype=np.float32)
39     if fps < 1:
40         print(f"Warning: Video has invalid FPS ({fps}). Proceeding, but timing might be off.")
41         # Use a default assumption if FPS is invalid but frames exist
42         fps = 30.0 # Or another sensible default
43
44     frames = []
45     extracted_count = 0
46     last_good_frame_processed = None # Store the last successfully processed frame
47
48     # Calculate indices of frames to attempt extraction (evenly spaced)
49     # Ensure indices are within the valid range [0, total_frames - 1]
50     indices = np.linspace(0, total_frames - 1, n_frames, dtype=int)

```

frame_slicer fig1

```

51
52 for i, frame_index in enumerate(indices):
53     cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
54     ret, frame = cap.read()
55
56     processed_frame = None
57     if ret and frame is not None:
58         try:
59             # Process valid frame
60             frame_resized = cv2.resize(frame, frame_size)
61             frame_rgb = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
62             processed_frame = frame_rgb.astype(np.float32) / 255.0
63             last_good_frame_processed = processed_frame # Update last good frame
64             extracted_count += 1
65         except cv2.error as e:
66             print(f"Warning: OpenCV error processing frame {frame_index}: {e}")
67             # Fallback to last good frame if available
68             if last_good_frame_processed is not None:
69                 processed_frame = last_good_frame_processed.copy()
70             else: # If no good frame seen yet, create a placeholder
71                 processed_frame = np.zeros((*frame_size[:-1], 3), dtype=np.float32)
72         except Exception as e:
73             print(f"Warning: Unexpected error processing frame {frame_index}: {e}")
74             if last_good_frame_processed is not None:
75                 processed_frame = last_good_frame_processed.copy()
76             else:
77                 processed_frame = np.zeros((*frame_size[:-1], 3), dtype=np.float32)
78
79     else:
80         # Handle read failure (e.g., end of video reached early, corrupted frame)
81         print(f"Warning: Failed to read frame at index {frame_index}. Using fallback.")
82         if last_good_frame_processed is not None:
83             processed_frame = last_good_frame_processed.copy()
84         else:
85             # If read fails and no previous frame exists, use a zero frame
86             processed_frame = np.zeros((*frame_size[:-1], 3), dtype=np.float32)
87
88     frames.append(processed_frame)
89
90 cap.release()

```

frame_slicer fig2

```

100 if final_frames.shape[0] < n_frames:
101     print(f"Warning: Padding needed, final array shape {final_frames.shape} vs target {n_frames}")
102     if final_frames.shape[0] == 0: # If somehow array is empty
103         padding = np.zeros((n_frames, *frame_size[:-1], 3), dtype=np.float32)
104     else:
105         padding_needed = n_frames - final_frames.shape[0]
106         # Use the very last frame in the list (could be a fallback frame) for padding
107         last_frame_for_padding = final_frames[-1][np.newaxis, ...]
108         padding = np.repeat(last_frame_for_padding, padding_needed, axis=0)
109     final_frames = np.concatenate((final_frames, padding), axis=0)
110 elif final_frames.shape[0] > n_frames:
111     # Should not happen with linspace logic, but truncate if it does
112     print(f"Warning: More frames than expected ({final_frames.shape[0]}), truncating to {n_frames}")
113     final_frames = final_frames[:n_frames]
114
115 # Final check of output shape
116 if final_frames.shape != (n_frames, frame_size[1], frame_size[0], 3):
117     print(f"Error: Final frame array shape mismatch! Expected {(n_frames, frame_size[1], frame_size[0], 3)}, Got {final_frames.shape}")
118     # Attempt to reshape or return None/zeros? Returning zeros is safer.
119     return np.zeros((n_frames, *frame_size[:-1], 3), dtype=np.float32)
120
121
122
123 return final_frames
124
125

```

frame_slicer fig3

```

1  import os
2  import numpy as np
3  import cv2
4  import traceback
5  from collections import Counter
6  from sklearn.model_selection import train_test_split
7  from tensorflow.keras.utils import Sequence
8  from tensorflow.keras.models import Sequential, load_model
9  from tensorflow.keras.layers import Input, Conv3D, MaxPooling3D, Flatten, Dense, Dropout, BatchNormalization
10 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
11 import tensorflow as tf
12
13 # === CONFIG ===
14 DATA_DIR = "D:\\K_REPO\\ComV\\train"
15 N_FRAMES = 30
16 IMG_SIZE = (96, 96)
17 EPOCHS = 10
18 BATCH_SIZE = 14
19 CHECKPOINT_DIR = r"D:\\K_REPO\\ComV\\AI_made\\trainnig_output\\checkpoint"
20 RESUME_TRAINING = 1
21 MIN_REQUIRED_FRAMES = 10
22 OUTPUT_PATH = r"D:\\K_REPO\\ComV\\AI_made\\trainnig_output\\final_model_2.h5"
23 # Optimize OpenCV
24 cv2.setUseOptimized(True)
25 cv2.setNumThreads(8)
26
27 # === VIDEO DATA GENERATOR ===
28 class VideoDataGenerator(Sequence):
29     def __init__(self, video_paths, labels, batch_size, n_frames, img_size):
30         self.video_paths, self.labels = self._filter_invalid_videos(video_paths, labels)
31         self.batch_size = batch_size
32         self.n_frames = n_frames
33         self.img_size = img_size
34         self.indices = np.arange(len(self.video_paths))
35         print(f"[INFO] Final dataset size: {len(self.video_paths)} videos")
36
37     def _filter_invalid_videos(self, paths, labels):
38         valid_paths = []
39         valid_labels = []
40
41         for path, label in zip(paths, labels):
42             cap = cv2.VideoCapture(path)
43             if not cap.isOpened():
44                 print(f"[WARNING] Could not open video: {path}")
45                 continue
46
47             total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
48             cap.release()
49
50             if total_frames < MIN_REQUIRED_FRAMES:
51                 print(f"[WARNING] Skipping {path} - only {total_frames} frames (needs at least {MIN_REQUIRED_FRAMES})")
52                 continue
53
54             valid_paths.append(path)
55             valid_labels.append(label)
56
57         return valid_paths, valid_labels

```

Training.py fig1


```

50
59 def __len__(self):
60     return int(np.ceil(len(self.video_paths) / self.batch_size))
61
62 def __getitem__(self, index):
63     batch_indices = self.indices[index*self.batch_size:(index+1)*self.batch_size]
64     x, y = [], []
65
66     for i in batch_indices:
67         path = self.video_paths[i]
68         label = self.labels[i]
69         try:
70             frames = self._load_video_frames(path)
71             x.append(frames)
72             y.append(label)
73         except Exception as e:
74             print(f"[WARNING] Error processing {path} - {str(e)}")
75             x.append(np.zeros((self.n_frames, *self.img_size, 3)))
76             y.append(label)
77
78     return np.array(x), np.array(y)
79
80 def _load_video_frames(self, path):
81     cap = cv2.VideoCapture(path)
82     total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
83
84     if total_frames < self.n_frames:
85         frame_indices = np.linspace(0, total_frames - 1, min(total_frames, self.n_frames), dtype=np.int32)
86     else:
87         frame_indices = np.linspace(0, total_frames - 1, self.n_frames, dtype=np.int32)
88
89     frames = []
90     for idx in frame_indices:
91         cap.set(cv2.CAP_PROP_POS_FRAMES, idx)
92         ret, frame = cap.read()
93         if not ret:
94             frame = np.zeros((*self.img_size, 3), dtype=np.uint8)
95         else:
96             frame = cv2.resize(frame, self.img_size)
97             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
98             frames.append(frame)
99
100     cap.release()
101
102     while len(frames) < self.n_frames:
103         frames.append(frames[-1] if frames else np.zeros((*self.img_size, 3), dtype=np.uint8))
104
105     return np.array(frames) / 255.0
106
107 def on_epoch_end(self):
108     np.random.shuffle(self.indices)

```

Training.py fig2

```

109
110 def create_model():
111     model = Sequential([
112         Input(shape=(N_FRAMES, *IMG_SIZE, 3)),
113         Conv3D(32, kernel_size=(3, 3, 3), activation='relu', padding='same'),
114         MaxPooling3D(pool_size=(1, 2, 2)),
115         BatchNormalization(),
116
117         Conv3D(64, kernel_size=(3, 3, 3), activation='relu', padding='same'),
118         MaxPooling3D(pool_size=(1, 2, 2)),
119         BatchNormalization(),
120
121         Conv3D(128, kernel_size=(3, 3, 3), activation='relu', padding='same'),
122         MaxPooling3D(pool_size=(2, 2, 2)),
123         BatchNormalization(),
124
125         Flatten(),
126         Dense(256, activation='relu'),
127         Dropout(0.5),
128         Dense(1, activation='sigmoid')
129     ])
130
131     model.compile(optimizer='adam',
132                   loss='binary_crossentropy',
133                   metrics=['accuracy'])
134
135     return model
136
137 def load_data():
138     video_paths, labels = [], []
139     for label_name in ["Fighting", "Normal"]:
140         label_dir = os.path.join(DATA_DIR, label_name)
141         if not os.path.isdir(label_dir):
142             raise FileNotFoundError(f"Directory not found: {label_dir}")
143
144         label = 1 if label_name.lower() == "fighting" else 0
145
146         for file in os.listdir(label_dir):
147             if file.lower().endswith((".mp4", ".mpeg", ".avi", ".mov")):
148                 full_path = os.path.join(label_dir, file)
149                 video_paths.append(full_path)
150                 labels.append(label)
151
152     if not video_paths:
153         raise ValueError(f"No videos found in {DATA_DIR}")
154
155     print(f"[INFO] Total videos: {len(video_paths)} (Fighting: {labels.count(1)}, Normal: {labels.count(0)})")
156
157     if len(set(labels)) > 1:
158         return train_test_split(video_paths, labels, test_size=0.2, stratify=labels, random_state=42)
159     else:
160         print("[WARNING] Only one class found. Splitting without stratification.")
161         return train_test_split(video_paths, labels, test_size=0.2, random_state=42)
162
163 def get_latest_checkpoint():
164     if not os.path.exists(CHECKPOINT_DIR):
165         os.makedirs(CHECKPOINT_DIR)
166         return None
167
168     checkpoints = [f for f in os.listdir(CHECKPOINT_DIR)
169                    if f.startswith('ckpt_') and f.endswith('.h5')]
170     if not checkpoints:
171         return None
172
173     checkpoints.sort(key=lambda x: int(x.split('_')[4].split('.')[0]))
174     return os.path.join(CHECKPOINT_DIR, checkpoints[-1])

```

Training.py fig3


```

176 def main():
177     # Load and split data
178     try:
179         train_paths, val_paths, train_labels, val_labels = load_data()
180     except Exception as e:
181         print(f"[ERROR] Failed to load data: {str(e)}")
182         return
183
184     # Create data generators
185     try:
186         train_gen = VideoDataGenerator(train_paths, train_labels, BATCH_SIZE, N_FRAMES, IMG_SIZE)
187         val_gen = VideoDataGenerator(val_paths, val_labels, BATCH_SIZE, N_FRAMES, IMG_SIZE)
188     except Exception as e:
189         print(f"[ERROR] Failed to create data generators: {str(e)}")
190         return
191
192     # Callbacks
193     callbacks = [
194         ModelCheckpoint(
195             os.path.join(CHECKPOINT_DIR, 'ckpt_{epoch}.h5'),
196             save_best_only=False,
197             save_weights_only=False
198         ),
199         CSVLogger('training_log.csv', append=True),
200         EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
201     ]
202
203     # Handle resume training
204     initial_epoch = 0
205     try:
206         if RESUME_TRAINING:
207             ckpt = get_latest_checkpoint()
208             if ckpt:
209                 print(f"[INFO] Resuming training from checkpoint: {ckpt}")
210                 model = load_model(ckpt)
211                 initial_epoch = int(ckpt.split('_')[1].split('.')[0])
212             else:
213                 print(f"[INFO] No checkpoint found, starting new training")
214                 model = create_model()
215             else:
216                 model = create_model()
217     except Exception as e:
218         print(f"[ERROR] Failed to initialize model: {str(e)}")
219         return
220
221     # Display model summary
222     model.summary()
223
224     # Train model
225     try:
226         print(f"[INFO] Starting training...")
227         history = model.fit(
228             train_gen,
229             validation_data=val_gen,
230             epochs=EPOCHS,
231             initial_epoch=initial_epoch,
232             callbacks=callbacks,
233             verbose=1
234         )
235     except Exception as e:
236         print(f"[ERROR] Training failed: {str(e)}")
237         traceback.print_exc()
238     finally:
239         model.save(OUTPUT_PATH)
240         print(f"[INFO] Training completed. Model saved to final_model_2.h5")
241
242 if __name__ == "__main__":
243     print(f"[INFO] Starting script...")
244     main()
245     print(f"[INFO] Script execution completed.")

```

Training.py fig4

```

1
2 import tensorflow as tf
3 from frame_slicer import extract_video_frames
4 import cv2
5 import os
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 # Configuration
10 import os
11 MODEL_PATH = os.path.join(os.path.dirname(__file__), "final_model_2.h5")
12 N_FRAMES = 30
13 IMG_SIZE = (96, 96)
14 # Define RESULT_PATH relative to the script location
15 RESULT_PATH = os.path.join(os.path.dirname(__file__), "results")
16
17 def fight_detec(video_path: str, debug: bool = True):
18     """Detects fight in a video and returns the result string and raw prediction score."""
19
20     class FightDetector:
21         def __init__(self):
22             self.model = self._load_model()
23
24         def _load_model(self):
25             # Ensure the model path exists before loading
26             if not os.path.exists(MODEL_PATH):
27                 print(f"Error: Model file not found at {MODEL_PATH}")
28                 return None
29             try:
30                 # Load model with compile=False if optimizer state isn't needed for inference
31                 model = tf.keras.models.load_model(MODEL_PATH, compile=False)
32                 if debug:
33                     print("\nModel loaded successfully. Input shape:", model.input_shape)
34                 return model
35             except Exception as e:
36                 print(f"Model loading failed: {e}")
37                 return None
38
39         def _extract_frames(self, video_path):
40             frames = extract_video_frames(video_path, N_FRAMES, IMG_SIZE)
41             if frames is None:
42                 print(f"Frame extraction returned None for {video_path}")
43                 return None
44
45             if debug:
46                 blank_frames = np.all(frames == 0, axis=(1, 2, 3)).sum()
47                 if blank_frames > 0:
48                     print(f"Warning: {blank_frames} blank frames detected")
49                 # Save a sample frame for debugging only if debug is True
50                 if frames.shape[0] > 0 and not np.all(frames[0] == 0): # Avoid saving blank frame
51                     sample_frame = (frames[0] * 255).astype(np.uint8)
52                     try:
53                         os.makedirs(RESULT_PATH, exist_ok=True) # Ensure result path exists
54                         debug_frame_path = os.path.join(RESULT_PATH, 'debug_frame.jpg')
55                         cv2.imwrite(debug_frame_path, cv2.cvtColor(sample_frame, cv2.COLOR_RGB2BGR))
56                         print(f"Debug frame saved to {debug_frame_path}")
57                     except Exception as e:
58                         print(f"Failed to save debug frame: {e}")
59                 else:
60                     print("Skipping debug frame save (first frame blank or no frames).")
61
62             return frames
63
64     return frames
65
66

```

Fight_detec_func.py fig1

```

65     def predict(self, video_path):
66         if not os.path.exists(video_path):
67             print(f"Error: Video not found at {video_path}")
68             return "Error: Video not found", None
69
70         try:
71             frames = self._extract_frames(video_path)
72             if frames is None:
73                 return "Error: Frame extraction failed", None
74
75             if frames.shape[0] != N_FRAMES:
76                 # Pad with last frame or zeros if not enough frames were extracted
77                 print(f"Warning: Expected {N_FRAMES} frames, got {frames.shape[0]}. Padding...")
78                 if frames.shape[0] == 0: # No frames at all
79                     frames = np.zeros((N_FRAMES, *IMG_SIZE, 3), dtype=np.float32)
80                 else: # Pad with the last available frame
81                     padding_needed = N_FRAMES - frames.shape[0]
82                     last_frame = frames[-1][np.newaxis, ...]
83                     padding = np.repeat(last_frame, padding_needed, axis=0)
84                     frames = np.concatenate((frames, padding), axis=0)
85                 print(f"Frames padded to shape: {frames.shape}")
86
87
88             if np.all(frames == 0):
89                 # Check if all frames are actually blank (can happen with padding)
90                 print("Error: All frames are blank after processing/padding.")
91                 return "Error: All frames are blank", None
92
93             # Perform prediction
94             prediction = self.model.predict(frames[np.newaxis, ...], verbose=0)[0][0]
95             # Determine result based on threshold
96             threshold = 0.61 # Example threshold
97             is_fight = prediction >= threshold
98             result = "FIGHT" if is_fight else "NORMAL"
99
100             # Calculate confidence (simple distance from threshold, scaled)
101             # Adjust scaling factor (e.g., 150) and base (e.g., 50) as needed
102             # Ensure confidence reflects certainty (higher for values far from threshold)
103             if is_fight:
104                 confidence = min(max((prediction - threshold) * 150 + 50, 0), 100)
105             else:
106                 confidence = min(max((threshold - prediction) * 150 + 50, 0), 100)
107
108             result_string = f"{result} ({confidence:.1f}% confidence)"
109
110
111             if debug:
112                 print(f"Raw Prediction Score: {prediction:.4f}")
113                 self._debug_visualization(frames, prediction, result_string, video_path)
114
115             return result_string, float(prediction) # Return string and raw score
116
117         except Exception as e:
118             print(f"Prediction error: {str(e)}")
119             # Consider logging the full traceback here in a real application
120             # import traceback
121             # print(traceback.format_exc())
122             return f"Prediction error: {str(e)}", None

```

Fight_detec_func.py fig2

```

123
124 def _debug_visualization(self, frames, score, result, video_path):
125     # This function will only run if debug=True is passed to fight_detec
126     print(f"\n--- Debug Visualization ---")
127     print(f"Prediction Score: {score:.4f}")
128     print(f"Decision: {result}")
129
130     # Avoid plotting if matplotlib is not available or causes issues in deployment
131     try:
132         import matplotlib.pyplot as plt
133         plt.figure(figsize=(15, 5))
134         num_frames_to_show = min(10, len(frames))
135         for i in range(num_frames_to_show):
136             plt.subplot(2, 5, i+1)
137             # Ensure frame values are valid for imshow (0-1 or 0-255)
138             img_display = frames[i]
139             if np.max(img_display) <= 1.0: # Assuming normalized float [0,1]
140                 img_display = (img_display * 255).astype(np.uint8)
141             else: # Assuming it might already be uint8 [0,255]
142                 img_display = img_display.astype(np.uint8)
143
144             plt.imshow(img_display)
145             plt.title(f"Frame {i}\nMean: {frames[i].mean():.2f}") # Use original frame for mean
146             plt.axis('off')
147         plt.suptitle(f"Video: {os.path.basename(video_path)}\nPrediction: {result} (Raw Score: {score:.4f}")
148         plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout
149
150         # Save the visualization
151         os.makedirs(RESULT_PATH, exist_ok=True) # Ensure result path exists again
152         base_name = os.path.splitext(os.path.basename(video_path))[0]
153         save_path = os.path.join(RESULT_PATH, f"{base_name}_prediction_result.png")
154         plt.savefig(save_path)
155         plt.close() # Close the plot to free memory
156         print(f"Debug visualization saved to: {save_path}")
157     except ImportError:
158         print("Matplotlib not found. Skipping debug visualization plot.")
159     except Exception as e:
160         print(f"Error during debug visualization: {e}")
161     print("--- End Debug Visualization ---")
162
163
164 # --- Main function logic ---
165 detector = FightDetector()
166 if detector.model is None:
167     # Model loading failed, return error
168     return "Error: Model loading failed", None
169
170 # Call the predict method
171 result_str, prediction_score = detector.predict(video_path)
172 return result_str, prediction_score

```

Fight_detec_func.py fig3

```

1
2 import cv2
3 import numpy as np
4 import os
5 from ultralytics import YOLO
6 import time
7 from typing import Tuple, Set, List
8
9 def detection(path: str) -> Tuple[Set[str], str]:
10     """
11     Detects and tracks objects in a video using YOLOv8 model, saving an annotated output video.
12
13     Args:
14         path (str): Path to the input video file. Supports common video formats (mp4, avi, etc.)
15
16     Returns:
17         Tuple[Set[str], str]:
18             - Set of unique detected object labels (e.g., {'Gun', 'Knife'})
19             - Path to the output annotated video with detection boxes and tracking IDs
20
21     Raises:
22         FileNotFoundError: If input video doesn't exist
23         ValueError: If video cannot be opened/processed or output dir cannot be created
24     """
25
26     # Validate input file exists
27     if not os.path.exists(path):
28         raise FileNotFoundError(f"Video file not found: {path}")
29
30     # --- Model Loading ---
31     # Construct path relative to this script file
32     model_path = os.path.join(os.path.dirname(__file__), "best.pt")
33     if not os.path.exists(model_path):
34         raise FileNotFoundError(f"YOLO model file not found at: {model_path}")
35
36     try:
37         model = YOLO(model_path)
38         class_names = model.names # Get class label mappings
39         print(f"[INFO] YOLO model loaded from {model_path}. Class names: {class_names}")
40     except Exception as e:
41         raise ValueError(f"Failed to load YOLO model: {e}")
42
43     # --- Output Path Setup ---
44     input_video_name = os.path.basename(path)
45     base_name = os.path.splitext(input_video_name)[0]
46     # Sanitize basename to prevent issues with weird characters in filenames
47     safe_base_name = "".join(c if c.isalnum() or c in ('-', '_') else '_' for c in base_name)
48
49     # Define output directory relative to this script
50     # In HF Spaces, this will be inside the container's file system
51     output_dir = os.path.join(os.path.dirname(__file__), "results")
52     temp_output_name = f"{safe_base_name}_output_temp.mp4"
53
54     try:
55         os.makedirs(output_dir, exist_ok=True) # Create output dir if needed
56         if not os.path.isdir(output_dir):
57             raise ValueError(f"Path exists but is not a directory: {output_dir}")
58     except OSError as e:
59         raise ValueError(f"Failed to create or access output directory '{output_dir}': {e}")
60
61     temp_output_path = os.path.join(output_dir, temp_output_name)
62     print(f"[INFO] Temporary output will be saved to: {temp_output_path}")
63
64

```

object_detect_yolo.py fig1

```

65 # --- Video Processing Setup ---
66 cap = cv2.VideoCapture(path)
67 if not cap.isOpened():
68     raise ValueError(f"Failed to open video file: {path}")
69
70 # Get video properties for output writer
71 # Use source FPS if available and reasonable, otherwise default to 30
72 source_fps = cap.get(cv2.CAP_PROP_FPS)
73 output_fps = source_fps if 10 <= source_fps <= 60 else 30.0
74
75 # Process at a fixed resolution for consistency or use source resolution
76 # Using fixed 640x640 as potentially used during training/fine-tuning
77 frame_width, frame_height = 640, 640
78 # OR use source resolution (might require adjusting YOLO parameters if model expects specific size)
79 # frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
80 # frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
81
82 try:
83     out = cv2.VideoWriter(
84         temp_output_path,
85         cv2.VideoWriter_fourcc(*'mp4v'), # Use MP4 codec
86         output_fps,
87         (frame_width, frame_height)
88     )
89     if not out.isOpened():
90         # Attempt alternative codec if mp4v fails (less common)
91         print("[WARNING] mp4v codec failed, trying avc1...")
92         out = cv2.VideoWriter(
93             temp_output_path,
94             cv2.VideoWriter_fourcc(*'avc1'),
95             output_fps,
96             (frame_width, frame_height)
97         )
98         if not out.isOpened():
99             raise ValueError("Failed to initialize VideoWriter with mp4v or avc1 codec.")
100
101 except Exception as e:
102     cap.release() # Release capture device before raising
103     raise ValueError(f"Failed to create VideoWriter: {e}")
104
105
106 # --- Main Processing Loop ---
107 detected_classes: List[str] = [] # Track detected object class names
108 start = time.time()
109 frame_count = 0
110 print(f"[INFO] Video processing started...")
111
112 while True:
113     ret, frame = cap.read()
114     if not ret: # End of video or read error
115         break
116
117     frame_count += 1
118     # Resize frame BEFORE passing to model
119     resized_frame = cv2.resize(frame, (frame_width, frame_height))
120
121     try:
122         # Run YOLOv8 detection and tracking on the resized frame
123         results = model.track(
124             source=resized_frame, # Use resized frame
125             conf=0.7, # Confidence threshold
126             persist=True, # Maintain track IDs across frames
127             verbose=False # Suppress Ultralytics console output per frame
128         )

```

object_detect_yolo.py fig2


```

130     # Check if results are valid and contain boxes
131     if results and results[0] and results[0].boxes:
132         # Annotate the RESIZED frame with bounding boxes and track IDs
133         annotated_frame = results[0].plot() # plot() draws on the source image
134
135         # Record detected class names for this frame
136         for box in results[0].boxes:
137             if box.cls is not None: # Check if class ID is present
138                 cls_id = int(box.cls[0]) # Get class index
139                 if 0 <= cls_id < len(class_names):
140                     detected_classes.append(class_names[cls_id])
141             else:
142                 print(f"[WARNING] Detected unknown class ID: {cls_id}")
143
144         # If no detections, use the original resized frame for the output video
145         annotated_frame = resized_frame
146
147         # Write the (potentially annotated) frame to the output video
148         out.write(annotated_frame)
149
150     except Exception as e:
151         print(f"[ERROR] Error processing frame {frame_count}: {e}")
152         # Write the unannotated frame to keep video timing consistent
153         out.write(resized_frame)
154
155
156     # --- Clean Up ---
157     end = time.time()
158     print(f"[INFO] Video processing finished. Processed {frame_count} frames.")
159     print(f"[INFO] Total processing time: {end - start:.2f} seconds")
160     cap.release()
161     out.release()
162     cv2.destroyAllWindows() # Close any OpenCV windows if they were opened
163
164
165     # --- Final Output Renaming ---
166     unique_detected_labels = set(detected_classes)
167     # Create a short string from labels for the filename
168     labels_str = "_".join(sorted(list(unique_detected_labels))).replace(" ", "_")
169     # Limit length to avoid overly long filenames
170     max_label_len = 50
171     if len(labels_str) > max_label_len:
172         labels_str = labels_str[:max_label_len] + "_etc"
173     if not labels_str: # Handle case where nothing was detected
174         labels_str = "no_detections"
175
176     final_output_name = f"{safe_base_name}_{labels_str}_output.mp4"
177     final_output_path = os.path.join(output_dir, final_output_name)
178
179     # Ensure final path doesn't already exist (rename might fail otherwise)
180     if os.path.exists(final_output_path):
181         os.remove(final_output_path)
182
183     try:
184         # Rename the temporary file to the final name
185         os.rename(temp_output_path, final_output_path)
186         print(f"[INFO] Detected object labels: {unique_detected_labels}")
187         print(f"[INFO] Annotated video saved successfully at: {final_output_path}")
188     except OSError as e:
189         print(f"[ERROR] Failed to rename {temp_output_path} to {final_output_path}: {e}")
190         # Fallback: return the temp path if rename fails but file exists
191         if os.path.exists(temp_output_path):
192             print(f"[WARNING] Returning path to temporary file: {temp_output_path}")
193             return unique_detected_labels, temp_output_path
194         else:
195             raise ValueError(f"Output video generation failed. No output file found.")
196
197
198     return unique_detected_labels, final_output_path

```

object_detect_yolo.py fig3

```

1  import cv2
2  import numpy as np
3  import os
4  from ultralytics import YOLO
5  import time
6  import tensorflow as tf
7  from frame_slicer import extract_video_frames
8  import matplotlib.pyplot as plt
9
10 from Fight_detec_func import fight_detec
11 from objec_detect_yolo import detection
12
13
14
15 # Entry point
16 path0 = input("Enter the local path : ")
17 path = path0.strip('"') # Remove extra quotes if copied from Windows
18 print(f"[INFO] Loading video: {path}")
19
20 fight_detec(path)
21 detection(path)
22

```

full_project.py fig1

11.3. Appendix C: Environment Setup

- `pip install -r requirements.txt`
- `python app.py`

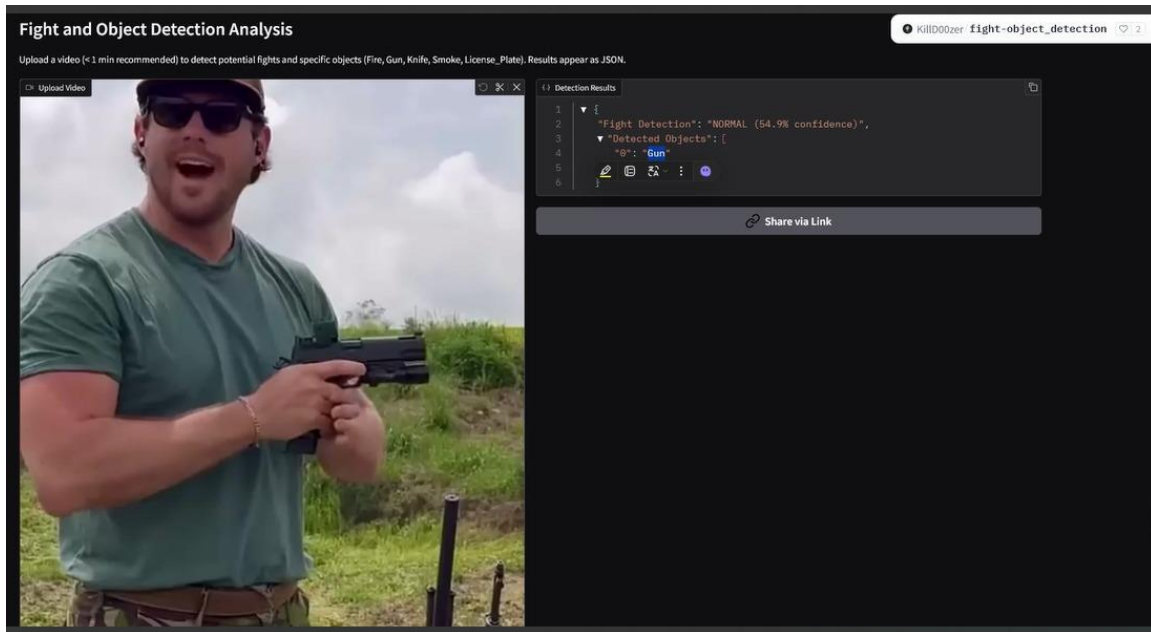
```

1  gradio>=3.0
2  tensorflow>=2.10 # Or specific version if needed, e.g., tensorflow==2.16.1
3  opencv-python>=4.6
4  ultralytics>=8.0
5  numpy>=1.22
6  matplotlib>=3.6 # Needed for debug plots in Fight_detec_func if debug=True

```

Requirements installation fig1

11.4. Appendix D: Model Output Samples



Model output fig1




Model output fig2

Fight and Object Detection Analysis

Upload a video (< 1 min recommended) to detect potential fights and specific objects (Fire, Gun, Knife, Smoke, License, Plate). Results appear as JSON.

Upload Video



Detection Results

```

1  {
2    "Fight Detection": "NORMAL (100.0% confidence)",
3    "Detected Objects": [
4      ]
5  }

```

Share via Link

Clear
Submit

Use via API - Built with Gradio - Settings

Model output fig3

11.5. Appendix E: GitHub/Hugging Face Link:

https://huggingface.co/spaces/KillD00zer/fight-object_detection