

Project Report

Deep Learning Model Analyzer

Master's in computer science

Student:	Rajesh Kumar Ramadas rajesh-kumar.ramadas@iu-study.org
Matriculation:	92125100
Course name	Project: Software Engineering (DLMCSPSE01)
University:	International University of Applied Sciences Juri-Gagarin-Ring 152 · D-99084 Erfurt
University Supervisor:	Dr. Holger Klus holger.klus@iu.org
Submission Date:	November 2024

Contents

1. PROJECT OVERVIEW	2
2. PROJECT GOALS	3
3. FEATURES	4
4. TECHNOLOGICAL STACK	5
5. PROJECT PLAN	6
6. UML FLOWCHART AND DIRECTORY	7
7. NON-FUNCTIONAL REQUIREMENT	8
8. PROJECT RISK	9
9. SOFTWARE ARCHITECTURE	9
10. TARGET USERS	11
11. LIMITATIONS	11
12. REPOSITORY & PREREQUISITES	12
13. STEPS TO USE APPLICATION	13
14. TOOL PERFORMANCE	18
15. CONCLUSION	19
16. BIBLIOGRAPHY	20

1. PROJECT OVERVIEW

The Deep Learning Model Analyzer is a tool designed to streamline the process of evaluating and analyzing deep learning models, particularly those based on the YOLOv8 architecture. This project aims to provide a comprehensive solution for researchers, data scientists, and developers who need to perform real-time video processing and model evaluation.

Efficiency: Streamlines the process of model evaluation, saving time and effort for researchers and developers.

Accessibility: The user-friendly GUI makes the tool accessible to a broader audience, including those with limited programming skills.

Flexibility: Supports various video sources and adjustable settings, catering to different evaluation needs and environments.

Real-Time Feedback: Provides immediate visual feedback on model performance, facilitating quick iterations and improvements.

The Deep Learning Model Analyzer is a robust and versatile tool designed to meet the evolving needs of the deep learning community. By combining real-time video processing with a user-friendly interface, it empowers users to efficiently evaluate and improve their models, driving advancements in the field of computer vision.

1.1. Core Components of Design

Main Application: The entry point of the application, responsible for initializing the GUI and managing the overall workflow.

GUI: Built using PyQt6, this component provides a user-friendly interface for interacting with the application. It includes features for loading models, adjusting video settings, and capturing snapshots.

Video Processing: Utilizes OpenCV to handle video streams, apply real-time predictions, and adjust video settings.

Model Handling: Integrates with the Ultralytics YOLOv8 library to load and evaluate models, ensuring compatibility with the latest advancements in object detection.

2. PROJECT GOALS

The Deep Learning Model Analyzer is designed with several key goals in mind to ensure it meets the needs of its users effectively. These goals focus on enhancing the usability, functionality, and performance of the tool, making it an asset for researchers, data scientists, and developers working with deep learning models.

2.1. Facilitate Model Evaluation

Objective: Provide a seamless process for loading and evaluating deep learning models, particularly YOLOv8 models.

Details: Users should be able to easily load their models into the application and run evaluations on various datasets. The tool should support different model formats and provide detailed performance metrics.

2.2. Enable Real-Time Video Processing

Objective: Allow users to process video streams in real-time and apply model predictions.

Details: The application should capture video from various sources (e.g., webcam, video files) and apply the loaded model to each frame in real-time. This enables users to see immediate results and make quick adjustments as needed.

2.3. Offer Adjustable Video Settings

Objective: Allow users to customize video settings to optimize the evaluation process.

Details: Users should be able to adjust parameters such as brightness, contrast, and frame rate directly from the GUI. These adjustments help in creating the best possible conditions for model evaluation and can be tailored to specific use cases.

2.4. Ensure Compatibility with YOLOv8

Objective: Maintain compatibility with the latest YOLOv8 models and annotation formats.

Details: The tool should support YOLOv8 models out-of-the-box, ensuring that users can leverage the latest advancements in object detection. It should also handle YOLOv8-compatible annotation formats for seamless integration.

2.5. Enable Snapshot Capture

Objective: Provide functionality for capturing and saving snapshots from video streams.

Details: Users should be able to capture snapshots at any point during video processing. These snapshots can be saved for further analysis, documentation, or reporting purposes.

3. FEATURES

- 3.1. Video Playback:** The application enables users to load, play, pause, and stop video files, providing flexible control over video playback for effective analysis. This feature supports common video formats and allows users to navigate frame-by-frame, ensuring precision during tasks like annotation and detection.
- 3.2. Camera Input:** Users can start and stop real-time video input from an attached camera. This feature allows for live analysis, enabling the YOLOv8 model and other processing tools to work directly on real-time data streams, making it ideal for applications requiring immediate feedback, such as surveillance or monitoring.
- 3.3. Edge Detection:** The tool includes three edge detection algorithms—Sobel, Canny, and Laplace of Gaussian:
- 3.4. Sobel:** A simple gradient-based algorithm ideal for highlighting vertical and horizontal edges.
- 3.5. Canny:** A multi-stage detector that reduces noise and preserves high-contrast edges, making it useful for precise boundary detection.
- 3.6. Laplace of Gaussian:** Combines Gaussian smoothing with Laplace filtering to detect edges and reduce noise simultaneously, providing refined edge outlines.
- 3.7. YOLOv8 Model Integration:** Users can load YOLOv8 models to perform object detection within video frames. The tool allows for filtering detections based on specific object classes and confidence thresholds, helping users to focus on relevant detections and achieve tailored results. This customization supports diverse applications, from tracking specific objects to analyzing motion in crowded scenes.
- 3.8. Brightness and Contrast Adjustment:** This feature enables users to modify the brightness and contrast of video frames, improving visibility and making details more discernible. These adjustments are particularly useful for enhancing low-light videos or emphasizing objects with low contrast relative to the background.
- 3.9. Cropping and Resizing:** Users can crop and resize video frames to focus on regions of interest or adjust dimensions for compatibility with other processing requirements. Cropping isolates specific parts of the frame, while resizing ensures consistency across datasets, making it easier to work with video in downstream processes.
- 3.10. Snapshot and Recording:** Users can capture snapshots of video frames at any moment, as well as save segments or entire video streams as recordings. Snapshots facilitate quick image export for further analysis or documentation, while recording provides a means of saving processed video segments for later review or reporting purposes.

4. TECHNOLOGICAL STACK

Tool leverages a robust technological stack:

4.1. **Programming Language: Python** (python , n.d.)

Chosen for its simplicity and strong compatibility with machine learning libraries, Python facilitates ease of development and integration with YOLOv8 models.

4.2. **GUI Framework: PyQt6**

PyQt6 allows for the creation of a cross-platform, user-friendly interface. Its extensive set of widgets enables smooth navigation and customization. (PyQt6, n.d.)

4.3. **Image Processing Libraries: OpenCV or PIL**

- **OpenCV:** Offers fast image manipulation functions ideal for large datasets. (opencv, n.d.)
- **PIL:** Provides a simpler alternative for basic image processing and format conversion tasks. (pillow, n.d.)

4.4. **Deep Learning: Ultralytics YOLOv8**

Ultralytics YOLOv8 is an advanced object detection model known for speed and accuracy, ideal for real-time applications. It integrates easily with Python, allowing for both training from scratch and fine-tuning on custom datasets. With pre-trained weights and comprehensive documentation, YOLOv8 streamlines deployment and adapts efficiently to various detection tasks.

5. PROJECT PLAN

5.1. Phase 1: Project Setup

- Set up the project directory structure.
- Create requirements.txt with necessary dependencies.
- Initialize a Git repository and create a .gitignore file.

5.2. Phase 2: Core Functionality Implementation

- Implement main.py to handle application initialization and version checks.
- Develop the VideoPlayer class in pyqt6_gui.py with the following methods:
- Implement the QTextEditLogger class for logging.

5.3. Phase 3: User Interface Development

- Design the main GUI layout in init_ui.
- Add functionality for loading videos, starting/stopping recording, and configuring settings.
- Integrate classes_to_detect for filtering and processing specific classes in the video.

5.4. Phase 4: Video Processing and Detection

- Implement video frame processing and object detection.
- Use classes_to_detect to filter detected objects.
- Display processed objects in the GUI.

5.5. Phase 5: Logging and Error Handling

- Integrate QTextEditLogger to display log messages in the GUI.
- Implement error handling for common issues (e.g., missing dependencies).

5.6. Phase 6: Documentation and Deployment

- Update README.md with detailed usage instructions and examples.
- Create additional documentation for developers (e.g., code comments, docstrings).
- Prepare the application for deployment (e.g., packaging, distribution).

6. UML FLOWCHART AND DIRECTORY



Fig 6.1: UML diagram for software

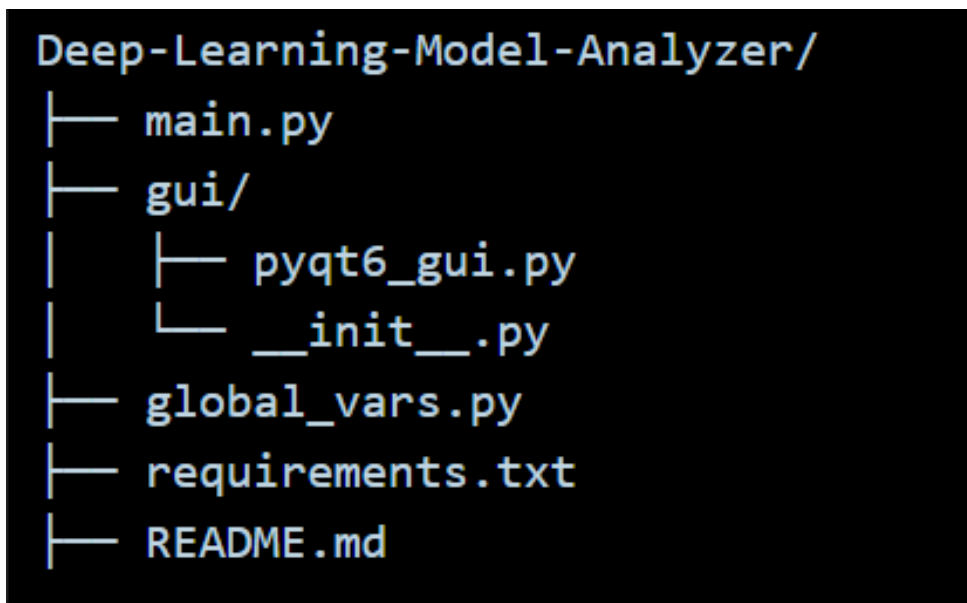


Fig 6.2: File structure for project development

7. NON-FUNCTIONAL REQUIREMENT

7.1. Performance

- **Startup Time:** The application should start within 5 seconds on a standard machine with the following specifications: Intel i5 processor, 8GB RAM, and SSD storage.
- **Frame Processing Rate:** The application should process video frames at a minimum rate of 30 frames per second (FPS) for videos with a resolution of 720p.
- **Latency:** The time taken to detect and display objects in a video frame should not exceed 100 milliseconds.

7.2. Reliability

- **Uptime:** The application should have an uptime of 99.9%, ensuring it is available for use most of the time.
- **Error Handling:** The application should gracefully handle errors such as missing dependencies, incorrect Python version, and unsupported video formats, providing meaningful error messages to the user.

7.3. Resource Utilization

- **CPU Usage:** The application should not exceed 70% CPU usage on a standard machine (Intel i5 processor, 8GB RAM) during normal operation.
- **Memory Usage:** The application should not exceed 2GB of RAM usage during normal operation.

7.4. Usability

- **User Interface:** The GUI should be intuitive and easy to navigate, with all major functions accessible within 3 clicks.
- **Documentation:** Comprehensive user documentation should be provided, including a user manual and in-app help.

7.5. Security

- **Data Protection:** The application should ensure that any user data, including video files and settings, are securely stored and not accessible to unauthorized users.
- **Authentication:** If the application supports remote access, it should include user authentication to prevent unauthorized access.

7.6. Maintainability

- **Modularity:** The application should be modular, allowing for easy updates and maintenance of individual components without affecting the entire system.

7.7. Compatibility

- **Operating Systems:** The application should be compatible with Windows 10 and later, macOS 10.15 and later, and major Linux distributions.
- **Python Version:** The application should run on Python 3.11.

8. PROJECT RISK

8.1. Technical Risks

- **Dependency Issues:** There is a risk that required Python packages may have compatibility issues or may not be available for certain operating systems.
- **Mitigation:** Regularly update and test dependencies. Use virtual environments to manage package versions.
- **Hardware Limitations:** The application may require more powerful hardware than available on some user machines, leading to poor performance.
- **Mitigation:** Provide clear hardware requirements and optimize the application to run efficiently on standard hardware.

8.2. Operational Risks

- **User Errors:** Users may encounter difficulties in using the application, leading to incorrect configurations or misuse.
- **Mitigation:** Provide comprehensive user documentation, in-app help, and tooltips for key features.
- **Data Loss:** There is a risk of data loss due to application crashes or improper handling of video files.
- **Mitigation:** Implement robust error handling and data backup mechanisms. Ensure the application can recover gracefully from crashes.

8.3. Project Management Risks

- **Scope Creep:** There is a risk of adding more features than initially planned, leading to delays and increased complexity.
- **Mitigation:** Clearly define the project scope and stick to it. Use a project management tool to track progress and manage changes.

9. SOFTWARE ARCHITECTURE

9.1. Main Components(main.py)

- Main Application
- Handles the initialization of the application.
- Checks for required Python version and packages.
- Launches the main GUI.

9.2. GUI (pyqt6_gui.py):

- Implements the main graphical user interface using PyQt6.
- Contains the VideoPlayer class which manages the video playback and user interactions.

9.3. Key Classes and Functions

- **VideoPlayer Class:**
 - **Attributes:**
 - `classes_to_detect`: List of classes to detect in the video.
 - `is_recording`: Boolean flag to indicate if recording is in progress.
 - `logger`: Logger for logging messages.
 - **Methods:**
 - `__init__`: Initializes the video player, sets up UI, video settings, timer, and logger.
 - `init_ui`: Sets up the user interface components.
 - `init_video_settings`: Initializes video-related settings.
 - `init_timer`: Sets up a timer for video playback or other timed operations.
 - `init_logger`: Initializes the logger for logging messages.
 - `stop_recording`: Stops the recording process and logs the action.
 - `adjust_window_size`: Adjusts the size of the application window.
 - **QTextEditLogger Class:**
 - Custom logging handler to display log messages in a QTextEdit widget.

9.4. Flow of Execution

- **Application Initialization:**
 - The application starts by checking the Python version and required packages.
 - If the checks pass, the main GUI is launched.
- **GUI Initialization:**
 - The VideoPlayer class is instantiated.
 - The `__init__` method sets up the window title, UI, video settings, timer, logger, and initializes `classes_to_detect`.
- **User Interactions:**
 - Users can interact with the GUI to load videos, start/stop recording, and configure settings.
 - The `classes_to_detect` list is used to filter and process specific classes in the video.
- **Video Processing:**
 - The application processes video frames, detects objects, and checks if they belong to `classes_to_detect`.
 - Detected objects are processed and displayed in the GUI.
- **Logging:**
 - Log messages are displayed in the GUI using the QTextEditLogger class

10. TARGET USERS

- 10.1. **Data Scientists and Machine Learning Engineers:** Professionals working with deep learning models who need an efficient, versatile tool to handle video data and run object detection or other model-based analyses.
- 10.2. **Researchers and Developers:** Individuals engaged in video processing, computer vision, and model evaluation who require reliable methods to assess and improve the accuracy and performance of detection models.
- 10.3. **Educators and Students:** Academics and learners in the fields of computer vision and deep learning who benefit from hands-on tools for exploring video processing and model application in real-world scenarios. This tool provides a practical framework to gain experience in these domains.

11. LIMITATIONS

11.1. Limited to YOLOv8 Models

Description: The current version of the tool is specifically designed to work with YOLOv8 models.

Impact: Users who wish to evaluate models based on other architectures (e.g., YOLOv5, Faster R-CNN, SSD) will not be able to use this tool without significant modifications.

11.2. Dependency on Specific Python Packages

Description: The tool relies on specific Python packages such as PyQt6 for the GUI and OpenCV for video processing.

Impact: Users must ensure that these dependencies are correctly installed and compatible with their system. Any issues with these packages can affect the functionality of the tool.

11.3. Performance Variability Based on Hardware

Description: The performance of the tool can vary significantly based on the hardware capabilities of the user's system.

Impact: Users with lower-end hardware may experience lag or reduced performance, especially when processing high-resolution video streams or large datasets.

11.4. Lack of Advanced Analytics and Visualization

Description: The current version of the tool focuses primarily on real-time video processing and model evaluation.

Impact: Users looking for advanced analytics and visualization tools (e.g., confusion matrices, precision-recall curves) will need to use additional software or tools.

12. REPOSITORY & PREREQUISITES

12.1. Repository: <https://github.com/RajeshRamadas/Deep-Learning-Model-Analyzer.git>

12.2. Prerequisites

Before installing the Yolo8 Annotation Tool, make sure you have the following installed on your system:

- **Python 3.9+:** Ensure Python is installed. You can download it from python.org.
- **pip:** Python package manager, usually installed with Python.

12.3. Installation Steps & Execution

12.3.1. Clone the repository:

```
git clone https://github.com/yourusername/deep-learning-model-analyzer.git
cd deep-learning-model-analyzer
```

12.3.2. Install the required dependencies:

```
pip install -r requirements.txt
```

12.3.1. Run the application:

```
python main.py
```



Fig 12.3.1: Model Analyzer

13. STEPS TO USE APPLICATION

13.1. Load a video:

Click on "Add Video MP4" to load a video file.

[Deep-Learning-Model-Analyzer/demo/video/sample1.mp4 at main · RajeshRamadas/Deep-Learning-Model-Analyzer · GitHub](#)

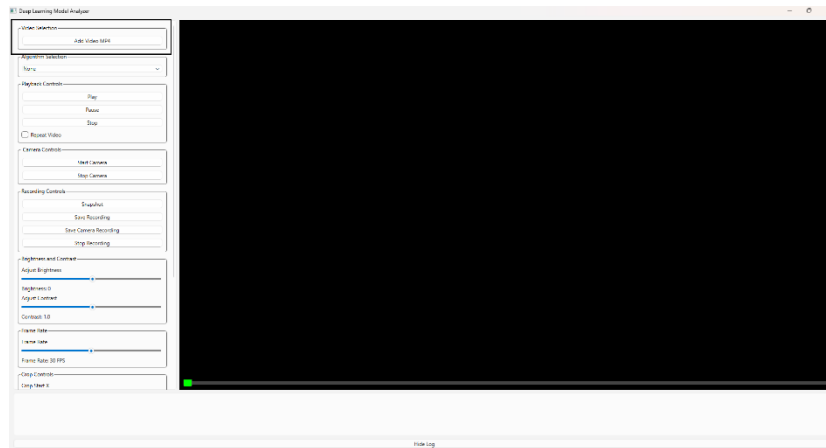


Fig 13.1: Add video for verification of model.

13.2. Select an algorithm:

Choose an edge detection algorithm or "Load YOLOv8 Model" from the "Algorithm Selection" dropdown.

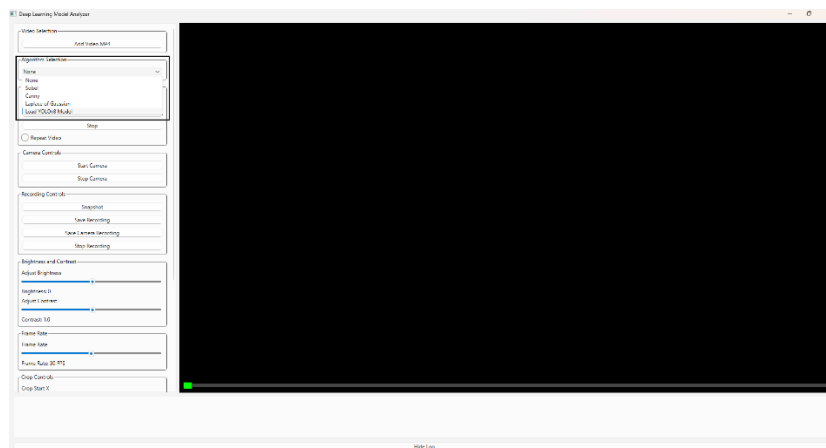


Fig 13.2: Select algo example [YOLO]

13.3. Load a YOLOv8 model:

If "Load YOLOv8 Model" is selected, click on "Load YOLOv8 Model" to load a model file. Sample default yolov8 model : [Deep-Learning-Model-Analyzer/demo/model/yolov8n.pt at main · RajeshRamadas/Deep-Learning-Model-Analyzer · GitHub](https://github.com/RajeshRamadas/Deep-Learning-Model-Analyzer/blob/main/demo/model/yolov8n.pt)

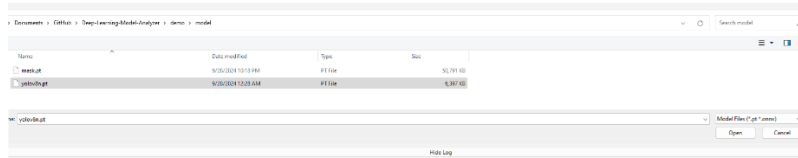


Fig 13.3.1: Select YOLO model.

Enter the classes to detect and the confidence threshold when prompted.

Yolo class ID: [Deep-Learning-Model-Analyzer/demo/model/yolo_classes.json at main · RajeshRamadas/Deep-Learning-Model-Analyzer](https://github.com/RajeshRamadas/Deep-Learning-Model-Analyzer/blob/main/demo/model/yolo_classes.json)

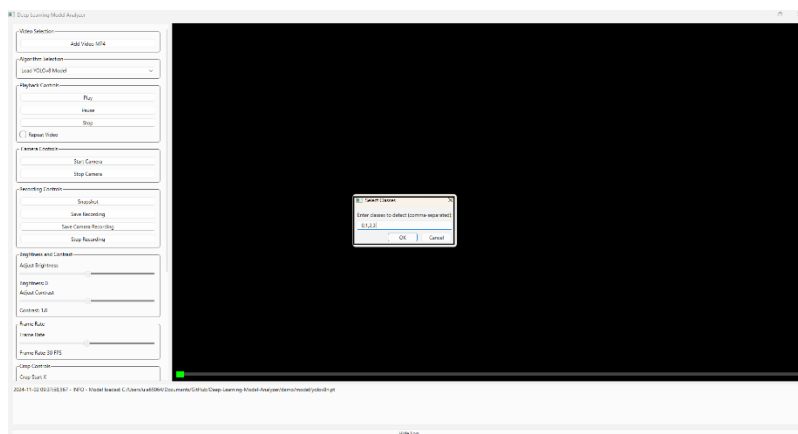


Fig 13.3.2: Add class ID for object detection with comma separator.

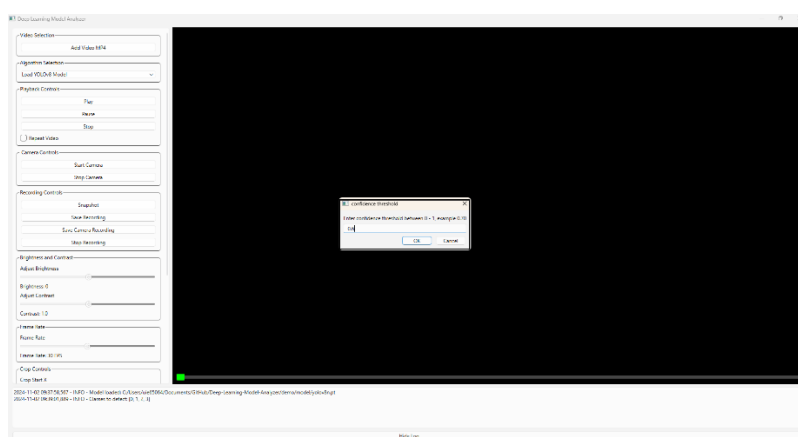


Fig 13.3.3: Confidence threshold between [0-1]

13.4. Adjust settings:

Use the sliders to adjust brightness, contrast, cropping, and resizing settings.

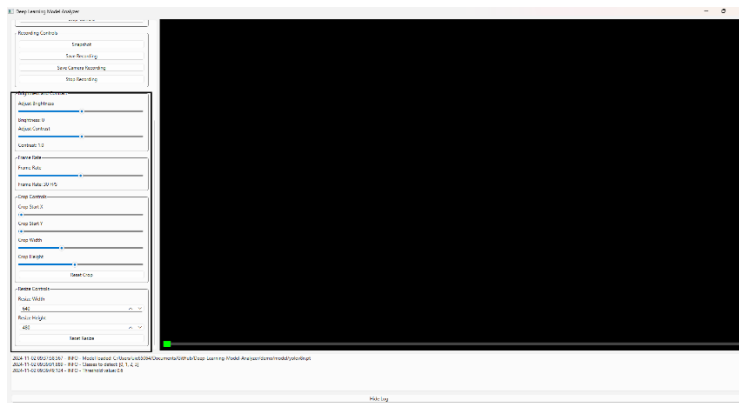


Fig 13.4.1: Adjust brightness, contrast, cropping.

13.5. Control playback:

Use the "Play", "Pause", and "Stop" buttons to control video playback.

Check "Repeat Video" to loop the video.

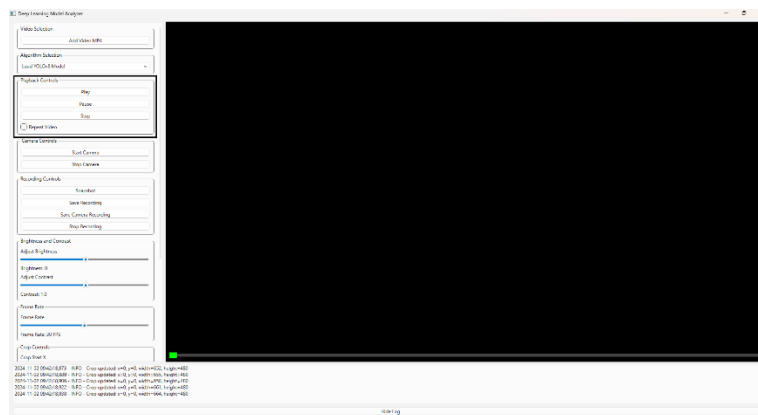


Fig 13.5.1: "Play", "Pause", and "Stop" buttons.

13.6. Capture snapshots and recordings:

Click on "Snapshot" to capture a snapshot of the current frame.

Click on "Save Recording" or "Save Camera Recording" to save video recordings.

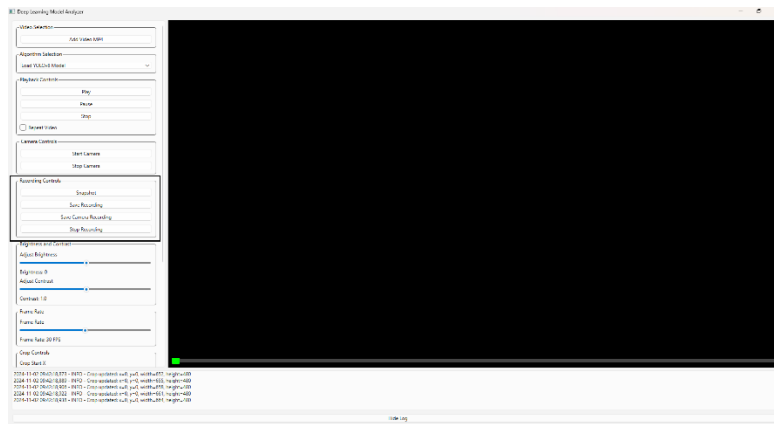


Fig 13.6.1: Recording option for snapshot and video.

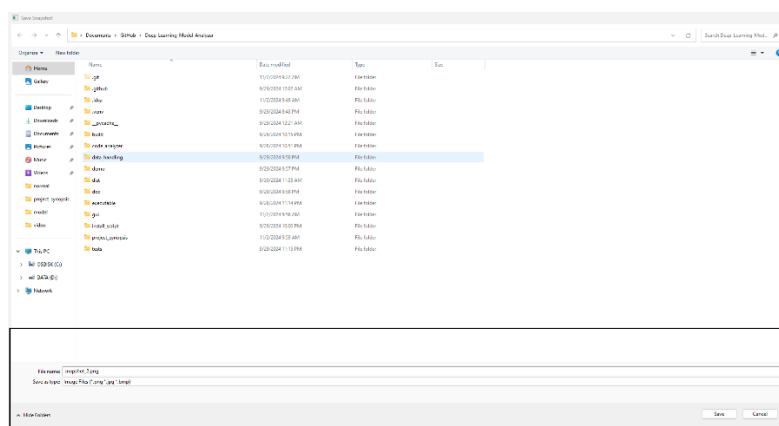


Fig 13.6.2: Snapshot save location.

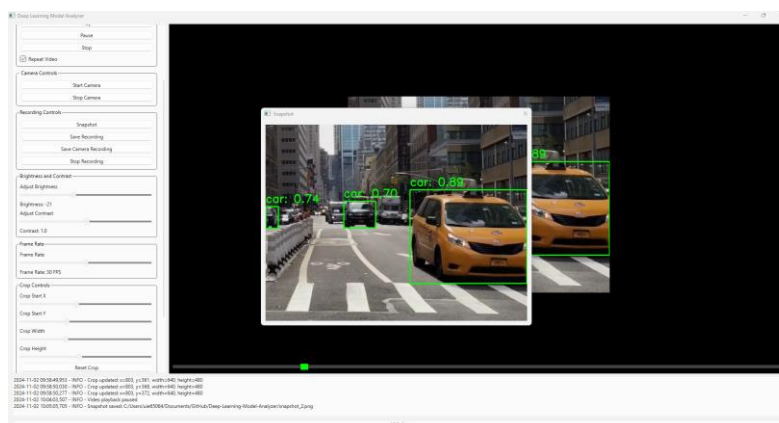


Fig 13.6.3: Saved snapshot pop-up.

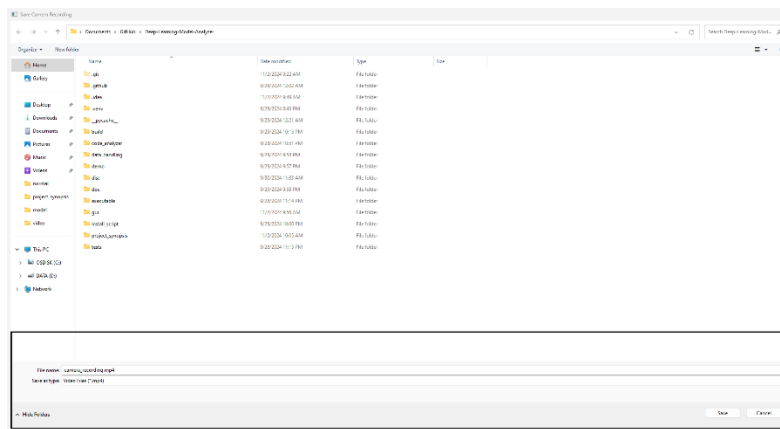


Fig 13.6.4: Video recording saving.

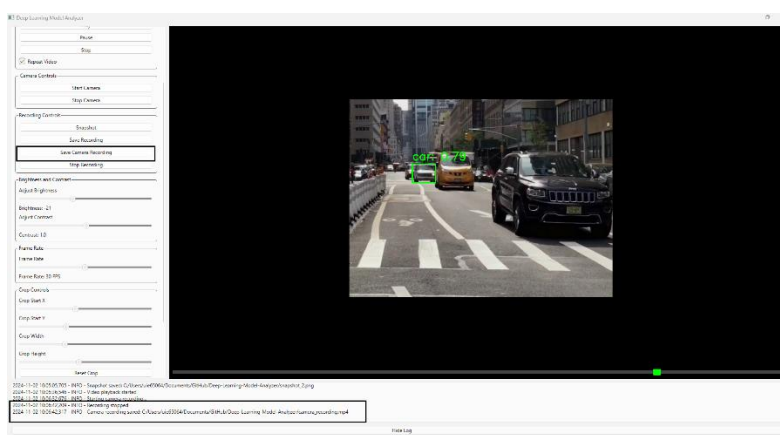


Fig 13.6.5: Recording saving log confirmation with path.

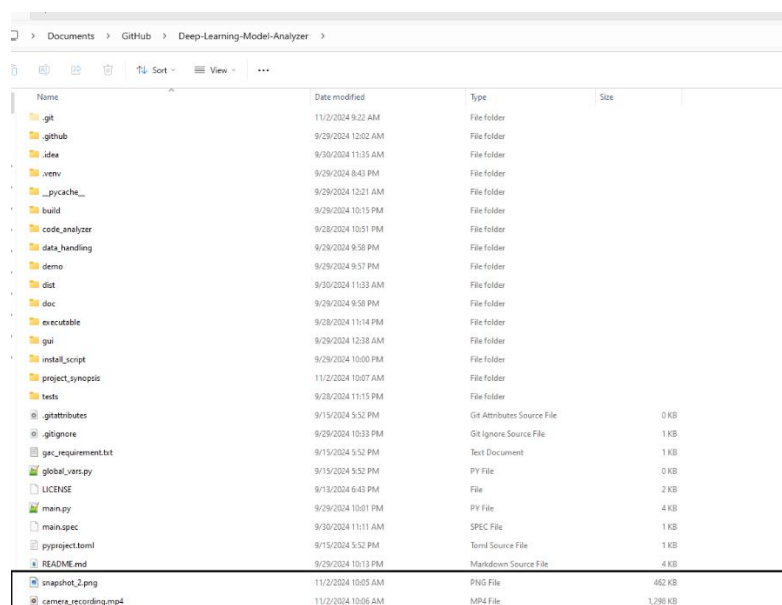


Fig 13.6.6: Saved recording and snapshot.

Sample Recording: [Deep-Learning-Model-Analyzer/demo/yolo8_video/camera_recording.mp4](#) at [main · RajeshRamadas/Deep-Learning-Model-Analyzer · GitHub](#)

14. TOOL PERFORMANCE

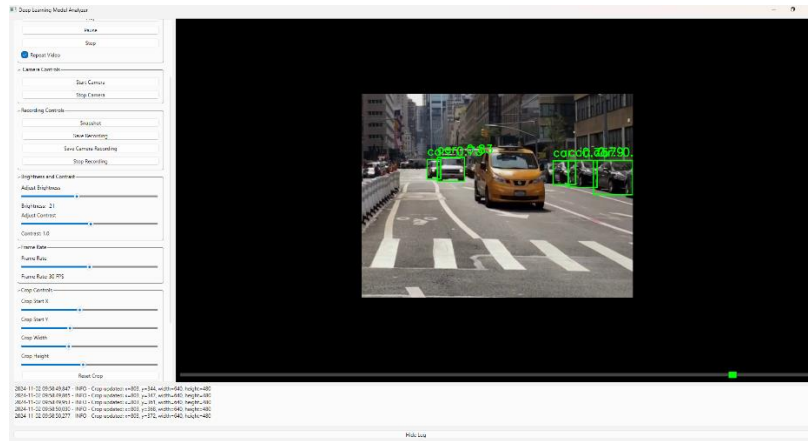


Fig 14.1: Object detection with bounding box.

14.1. Real-Time Processing

The tool leverages OpenCV for real-time video processing, enabling immediate application of YOLOv8 model predictions on video streams.

Users can see real-time results, which is essential for applications requiring instant feedback, such as surveillance, autonomous driving, and interactive systems.

14.2. Efficient Model Loading

The tool efficiently loads YOLOv8 models, ensuring minimal delay between model selection and evaluation.

Quick model loading enhances user experience by reducing wait times and allowing for rapid iterations and testing.

14.3. Adjustable Video Settings

Users can adjust video settings such as brightness, contrast, and frame rate to optimize the evaluation process.

Customizable settings help in creating the best possible conditions for model evaluation, improving the accuracy and relevance of the results.

15. CONCLUSION

The Deep Learning Model Analyzer is a robust and versatile tool designed to meet the needs of researchers, data scientists, and developers working with deep learning models, particularly those based on the YOLOv8 architecture. This project has been developed with a focus on providing a comprehensive solution for model evaluation, real-time video processing, and user-friendly interaction.

15.1. Key Achievements

- Model Evaluation:
- Real-Time Video Processing
- User-Friendly Interface
- Adjustable Video Settings
- Snapshot Capture
- Real-Time Feedback

15.2. Limitations and Future Enhancements

While the Deep Learning Model Analyzer offers significant benefits, it also has some

- Model Compatibility: Currently limited to YOLOv8 model.
- Dependency Management: Ensuring compatibility with specific Python packages.
- Performance Variability: The tool's performance based on hardware capabilities.
- Advanced Features: Adding advanced analytics, visualization tools, and cloud integration can enhance the tool's functionality and scalability.
- Unable to create exe due to dependency of yolov8 module.

15.3. Future Directions

To address these limitations and further enhance the tool, the following future directions are proposed:

- Extended Model Support: Expand compatibility to include other popular deep learning models and architectures, making the tool more versatile.
- Advanced Analytics and Visualization: Integrate advanced analytics and visualization features to provide deeper insights into model performance.

15.4. Final Thoughts

The Deep Learning Model Analyzer represents a significant step forward in the field of model evaluation and real-time video processing. By combining powerful functionalities with a user-friendly interface, it empowers users to efficiently evaluate and improve their deep learning models. As the tool continues to evolve, addressing its current limitations and incorporating user feedback will be crucial for its ongoing success and adoption.

In conclusion, the Deep Learning Model Analyzer is an asset for anyone working with deep learning models, offering a comprehensive and accessible solution for model evaluation and real-time video processing. With continued development and enhancement, it has the potential to become an indispensable tool in the deep learning community.

16. BIBLIOGRAPHY

- 16.1. opencv. (n.d.). Retrieved from <https://opencv.org/>: <https://opencv.org/>
- 16.2. pillow. (n.d.). Retrieved from <https://pypi.org>: <https://pypi.org/project/pillow/>
- 16.3. PyQt6. (n.d.). Retrieved from pypi.org: <https://pypi.org/project/PyQt6/>
- 16.4. python . (n.d.). Retrieved from <https://www.python.org/>: <https://www.python.org/>