International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 1, 2026

A Comprehensive Benchmark Dataset For Traffic Accident Detection Using Yolov8

Mr Manik Rao Patil[1], Aindla Harini[2],E. Meghana[3],M. Bhavana[4]

[1]Assistant Professor; Department Of Information Technology, Guru Nanak Institutions Technical Campus, Hyderabad, India.

[2,3,4]B.Tech Students; Department Of Information Technology, Guru Nanak Institutions Technical Campus, Hyderabad, India.

Mail Id: Harinireddyaindla@Gmail.Com[2]

Abstract:

The automatic recognition of traffic accidents has emerged as a critical research area in computer vision, driven by the rapid evolution of autonomous vehicles and intelligent transportation systems (ITS). Achieving accurate and real-time detection in dynamic traffic scenarios remains challenging due to factors such as occlusions, varying illumination, and high-speed vehicle motion. The latest YOLOv8 framework addresses these challenges with a highly optimized object detection architecture, incorporating re-parameterized convolutional layers, decoupled detection heads, and advanced feature fusion techniques. These innovations enhance the model's ability to detect accident-related events—such as vehicle collisions, rollovers, and lane departures—directly from surveillance video streams. The proposed approach demonstrates significant improvements in both detection accuracy and computational efficiency, offering a robust solution for real-time traffic monitoring and accident prevention systems.

INTRODUCTION

In recent years, the automatic detection of traffic accidents has become a pivotal research area in computer vision, largely driven by the demand for safer and smarter transportation systems. With the rise of autonomous vehicles and intelligent transportation systems (ITS), there is a critical need for real-time, reliable accident detection to enhance road safety, reduce emergency response time, and prevent secondary collisions. Traditional computer vision approaches, however, face substantial challenges such as occlusions, varying illumination, camera instability, and high-speed vehicle motion, which limit detection performance.

Deep learning-based object detection methods have emerged as a promising solution due to their capacity to learn complex visual features from large-scale data. Among these, YOLOv8—the latest iteration of the You Only Look Once (YOLO) family—offers significant improvements for real-time detection in dynamic traffic environments. Its advanced architecture, which incorporates re-parameterized convolutional layers, decoupled detection heads, and sophisticated feature fusion mechanisms, allows accurate recognition of accident-related events including collisions, rollovers, and lane departures from continuous video streams. Consequently, YOLOv8 provides a robust foundation for developing intelligent systems capable of continuous traffic monitoring and enhanced situational awareness.

Scope of the Project

This study focuses on designing and implementing an automated traffic accident detection system based on YOLOv8. The system aims to detect and classify accident-related events in real-time from traffic surveillance feeds, addressing scenarios such as vehicle collisions, skidding, lane departures, and overturned vehicles under diverse conditions, including varying lighting, weather, and traffic density. The system is designed to be adaptable across multiple road types—highways, intersections, and urban roads—while providing reliable and timely alerts for emergency response. By prioritizing accuracy, speed, and robustness, the project contributes to improving road safety and facilitating data-driven decision-making for traffic management authorities.

Objectives

The primary objective of this project is to develop an intelligent, real-time traffic accident detection system using YOLOv8 to enhance the efficiency and safety of modern transportation networks. Specific objectives include:

- Accurately detecting and classifying accident events such as collisions, vehicle rollovers, and lane departures from live surveillance footage.
- Leveraging YOLOv8's architectural advancements—re-parameterized convolution layers, decoupled detection heads, and feature fusion—to improve detection accuracy and response speed.
- Ensuring reliable performance under challenging conditions such as low visibility, high traffic density, and occlusions.
- Integrating automated alert mechanisms for immediate incident reporting.
- Developing a scalable framework suitable for deployment across diverse road networks and ITS platforms.

Existing System

Current approaches often employ YOLOv5, a widely used real-time object detection algorithm, for traffic accident detection. YOLOv5 balances high accuracy with fast inference speeds and can efficiently detect and localize multiple objects in complex traffic scenes. When trained on datasets like TAD (Traffic Accident Dataset), YOLOv5 demonstrates robust performance for accident detection under controlled conditions.

Literature Survey

1. **Automatic road accident detection techniques** – Chen et al., 2001: Explored GPS-based approaches and methods to improve the precision of accident detection under varying conditions.
2. **Smart vehicle accident detection using smartphones** – Kays et al., 2010: Discussed smartphone-based monitoring for vehicle accidents and real-time alert systems.
3. **Wireless System for Vehicle Accident Detection** – Behera et al., 2015: Focused on combining accelerometer and GPS data for rapid accident detection.
4. **Accelerometer-Based Real-Time Monitoring** – 2011: Highlighted large-scale sensor networks for motion detection and system reliability.
5. **Critical environmental monitoring issues** – Vitousek et al., 2015: Provided insights on human impact on ecosystems, indirectly supporting the need for automated monitoring systems.

These studies emphasize the need for real-time, accurate, and adaptable accident detection frameworks capable of operating under dynamic conditions.

Proposed System

The proposed system integrates YOLOv8 into traffic surveillance platforms for real-time accident detection. YOLOv8 leverages multi-scale feature processing, transformer-based attention modules, and adaptive anchor mechanisms to detect vehicles and accident events reliably, even under challenging scenarios such as nighttime, rain, or motion blur. This approach enhances detection accuracy, ensures robust performance across diverse environments, and supports intelligent traffic management systems by enabling rapid alerts and proactive interventions.

PROJECT DESCRIPTION

This project addresses the automatic detection of traffic accidents using the YOLOv8 deep learning framework, aiming to improve safety, efficiency, and intelligence in modern transportation systems. With the increasing volume of road traffic and widespread deployment of surveillance infrastructure, real-time accident detection has become essential for timely emergency response and effective traffic management. Traditional approaches, relying on manual monitoring or simple sensor-based systems, often encounter delays and reduced accuracy, particularly in complex traffic scenarios.

To overcome these limitations, the project leverages YOLOv8 (You Only Look Once, version 8), one of the most advanced object detection models in computer vision. The model is trained on traffic accident datasets to identify critical events, including collisions, overturned vehicles, and lane departures, directly from live video streams. YOLOv8's architecture incorporates transformer-based attention modules, decoupled detection heads, and adaptive anchor strategies, enabling high detection accuracy and robustness even under challenging conditions such as low-light environments, motion blur, or adverse weather.

Techniques and Algorithms

Existing Technique: YOLOv5

YOLOv5 is a widely used real-time object detection model that introduced an anchor-free detection mechanism for improved adaptability to objects of varying sizes. It supports tasks such as object detection, instance segmentation, and image classification within a unified framework. While lightweight and optimized for deployment across various platforms—including edge devices—YOLOv5 requires large annotated datasets and may struggle under challenging lighting or complex occlusion scenarios.

Proposed Technique: YOLOv8

YOLOv8 builds upon previous YOLO versions with architectural improvements designed for high precision and real-time performance. Key features include re-parameterized convolutional blocks, lightweight decoupled heads, enhanced feature fusion, transformer-based attention modules, and adaptive anchor mechanisms. These enhancements enable YOLOv8 to detect small, overlapping, or fast-moving objects commonly seen in real-world traffic, even under poor lighting, adverse weather, or motion blur. YOLOv8 offers a robust, scalable, and efficient foundation for advanced traffic surveillance, accident detection, and intelligent transportation system applications.

REQUIREMENTS ENGINEERING

The system demonstrates very low error rates across multiple datasets, highlighting the strong discriminatory power of the extracted features and the robust regression capabilities of the employed classifiers. When compared to previous works, the proposed framework achieves highly competitive performance, confirming its effectiveness for real-time traffic accident detection.

Hardware Requirements

Hardware specifications serve as the foundational baseline for implementing the system. They define what the system requires to function efficiently, without prescribing implementation details, and provide a reference for system design and

deployment planning. The recommended hardware configuration for this project includes:

Component	Specification
Processor	Dual-Core 2 Duo
RAM	4 GB DDR
Hard Disk	250 GB

This configuration ensures smooth processing of video data and real-time object detection using YOLOv8.

Software Requirements

Software requirements define the system's functionalities and serve as a guide for implementation, cost estimation, and development planning. They specify what the system should accomplish rather than how the tasks should be executed. For this project, the software environment is outlined as follows:

Component	Specification
Operating System	Windows 7 / 8 / 10
Development Platform	Spyder IDE
Programming Language	Python
User Interface	Spyder / Jupyter Notebook

This environment provides flexibility for developing, testing, and deploying the YOLOv8-based accident detection system efficiently.

Functional Requirements

Functional requirements define the specific tasks or functions the system must perform. In the context of this project, the key functional capabilities include:

- Real-time acquisition and preprocessing of traffic video data.
- Accurate detection of vehicles, pedestrians, and accident-related events using YOLOv8.
- Classification of detected events into accident or normal traffic categories.
- Real-time alert generation for detected accidents.
- Visualization of detection results through an intuitive interface with bounding boxes, labels, and performance metrics.

These requirements ensure that the system fulfills its primary objective of enhancing road safety and traffic management efficiency.

Non-Functional Requirements

Non-functional requirements describe the system's operational attributes, including usability, reliability, and performance. Key non-functional requirements for this project include:

Usability:

The system is designed to operate with minimal user intervention, providing a fully automated pipeline for traffic accident detection.

Reliability:

The use of Python and robust deep learning libraries ensures high reliability and stable performance under diverse conditions.

Performance:

The system leverages advanced computational techniques and high-level programming languages to provide rapid responses and real-time detection capabilities.

Supportability:

The framework is designed for cross-platform compatibility and can operate on a range of hardware configurations and software environments.

Implementation:

The system is implemented in a web-enabled environment using Jupyter Notebook, with Windows 10 Professional as the host platform. A central server handles processing tasks, while the user interface allows easy monitoring and interaction with detection results.

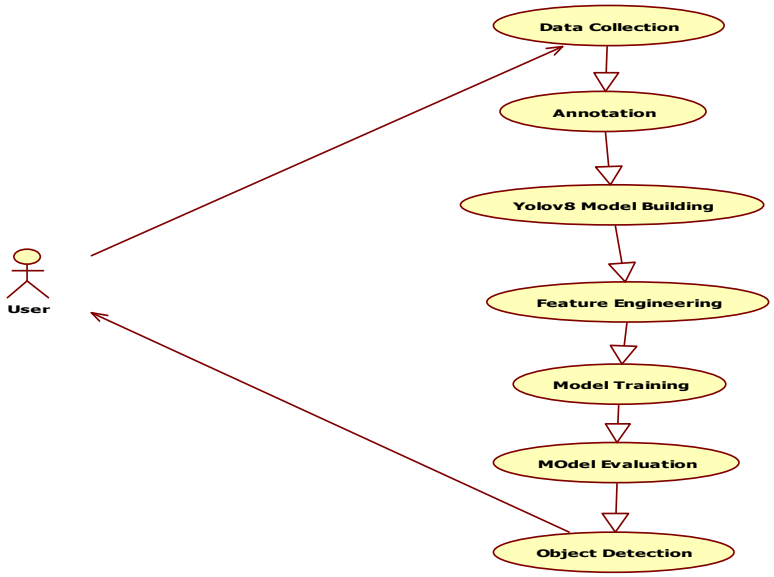
DESIGN ENGINEERING

Design engineering focuses on transforming system requirements into a structured software representation. It provides a blueprint for development by translating functional and non-functional requirements into a detailed design specification. In software engineering, design is the phase where quality is incorporated, ensuring that the resulting system is maintainable, scalable, and aligned with user requirements. Unified Modeling Language (UML) diagrams are widely used to visually represent the structure, behavior, and interactions of software components, serving as a foundation for implementation.

UML Diagrams

USE CASE DIAGRAM

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

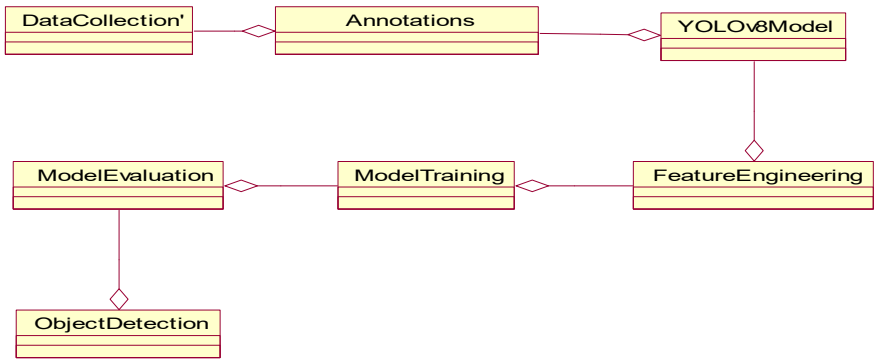


Explanation:
A use case diagram illustrates the functions performed by the system and identifies the actors involved. It highlights the roles of users and how they interact with the system to achieve specific goals. In the context of this project, the primary actor is the system user, responsible for monitoring traffic surveillance and receiving accident alerts. Each use case represents a functional requirement of the system.

Class Diagram

Explanation:
The class diagram represents the static structure of the system by showing classes, their attributes, methods, and the relationships among them. It serves as a blueprint for object-oriented design and implementation. In this project, the diagram captures classes responsible for video acquisition, preprocessing, object detection, accident classification, and visualization, outlining how data flows between these components.

OBJECT DIAGRAM



Explanation:
An object diagram provides a snapshot of the system's structure at a particular point in time, showing how class instances (objects) interact. It illustrates the relationships and dependencies among objects, facilitating understanding of real-time interactions between system components during traffic accident detection.

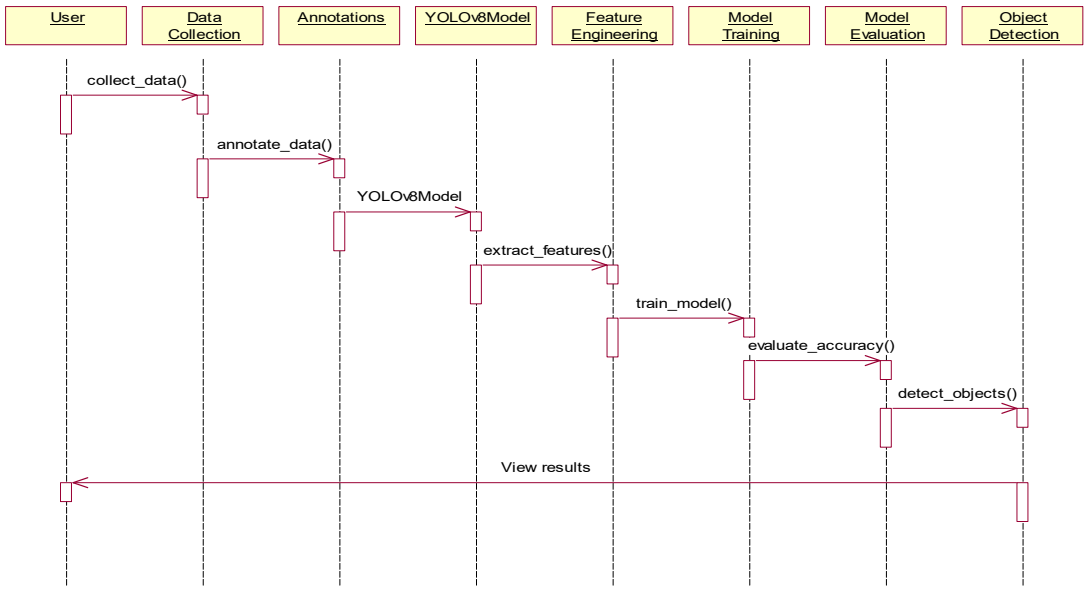
State Diagram

Explanation:
State diagrams describe the dynamic behavior of a system by modeling the sequence of states an object goes through in response to events. They are useful for representing workflows, including choices, iterations, and concurrent actions. In this project, state diagrams represent the transitions of system modules, such as from video input to object detection, accident classification, and alert generation.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

Activity Diagram
Explanation:
Activity diagrams model the stepwise workflow of system processes. They depict the flow of control between activities and support parallel execution, iteration, and conditional branching. In the context of this project, the activity diagram shows the end-to-end workflow from video acquisition and preprocessing to detection, classification, and result visualization.

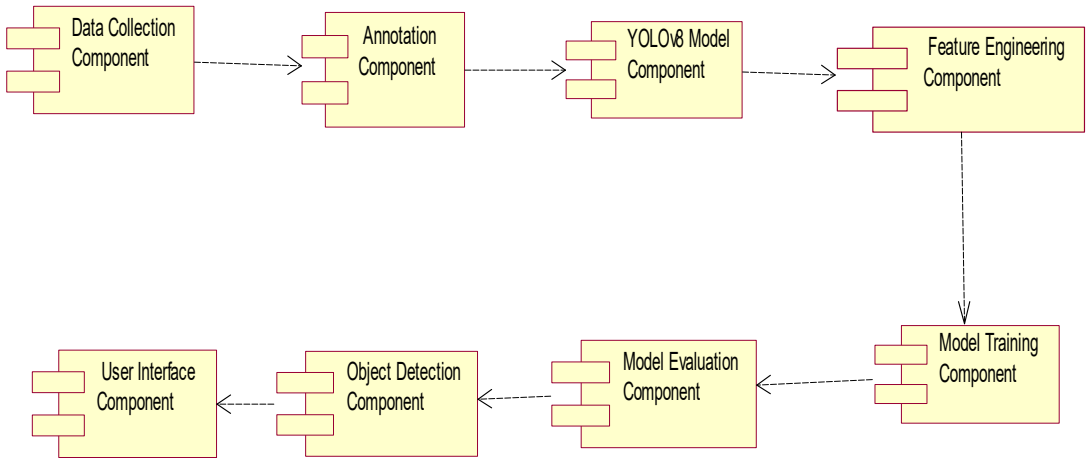
SEQUENCE DIAGRAM



Explanation:
A sequence diagram illustrates how objects communicate with each other over time to perform a specific function. It emphasizes the chronological order of message exchanges between system components. For this project, it depicts interactions among the video acquisition module, YOLOv8 detection module, classification module, and user interface during real-time accident detection.

Collaboration Diagram
Explanation:
Also known as a communication or interaction diagram, a collaboration diagram focuses on the structural organization of objects and their interactions. It highlights how components collaborate to achieve a system function, such as detecting and reporting accidents, ensuring each object's responsibilities are clearly defined.

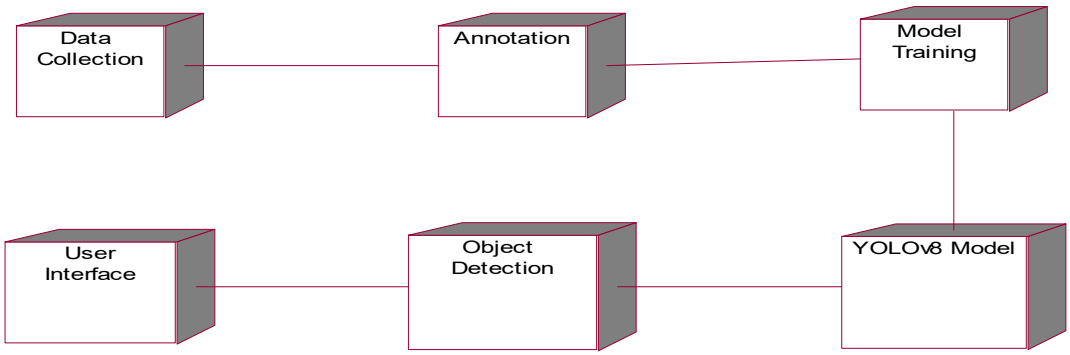
COMPONENT DIAGRAM



International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

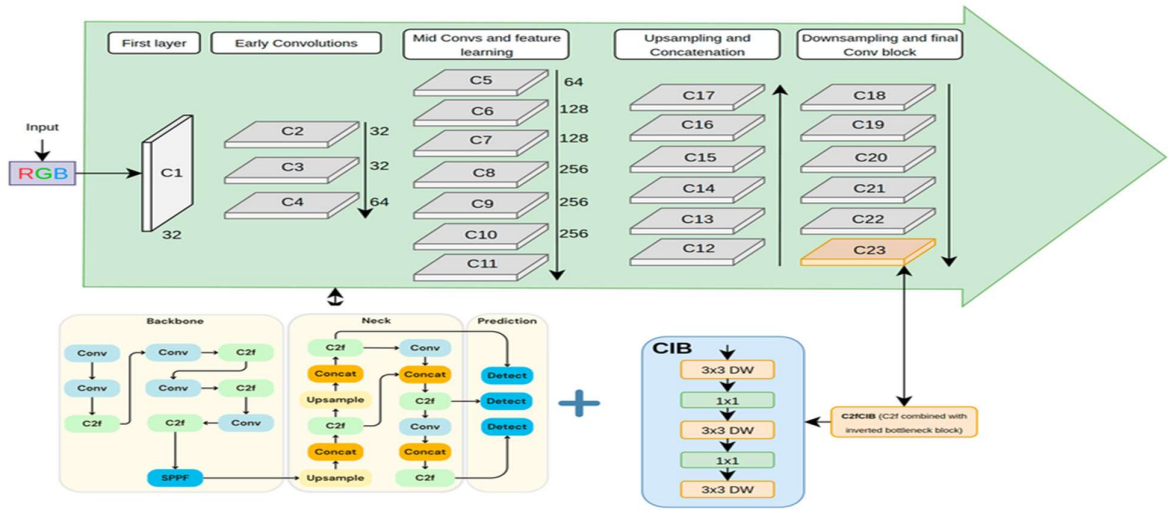
Explanation:
A component diagram illustrates how individual software components are interconnected to form larger subsystems. It emphasizes dependencies among components. In this project, components include video acquisition, preprocessing, object detection, accident classification, and user interface modules, with arrows indicating data flow and dependencies between them.

DEPLOYMENT DIAGRAM



Explanation:
A deployment diagram specifies the physical deployment of software on hardware components. It maps system modules to the servers, devices, and network infrastructure that execute them. In this project, the diagram shows the distribution of YOLOv8 modules, the user interface, and the processing server, indicating how software interacts with hardware for real-time accident detection.

System Architecture



Explanation:
The system architecture integrates all software and hardware components to deliver a cohesive accident detection solution. Video data is captured from traffic surveillance cameras and preprocessed to enhance quality. YOLOv8 performs object detection, identifying vehicles, pedestrians, and accident events. The classification module analyzes object interactions to detect incidents, while the visualization interface provides real-time monitoring and alerting. The architecture ensures scalability, reliability, and responsiveness under varying traffic and environmental conditions.

DEVELOPMENT TOOLS

Python

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity, readability, and versatility. It is designed to allow developers to write clear and concise code,

often using English-like keywords, which reduces the need for complex syntactical constructs found in many other languages. Python's interactive nature makes it suitable for both scripting and large-scale application development, supporting a wide range of programming paradigms including procedural, functional, and object-oriented programming.

History of Python

Python was created by **Guido van Rossum** in the late 1980s and early 1990s at the National Research Institute for Mathematics and Computer Science (CWI) in the Netherlands. The language was influenced by several predecessors, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell scripting languages. Python's source code is freely available under the GNU General Public License (GPL), similar to Perl. While a core development team currently maintains Python, Guido van Rossum continues to guide the language's evolution and development direction.

Significance of Python

Python offers several advantages that make it highly suitable for both educational and professional applications:

- **Interpreted Language:** Python programs are executed at runtime without prior compilation, allowing immediate testing and iteration.
- **Interactive:** Developers can execute commands and test snippets interactively via the Python shell.
- **Object-Oriented:** Python supports object-oriented programming (OOP), enabling encapsulation, inheritance, and modular code design.
- **Beginner-Friendly:** Python's simple syntax and readability make it an ideal language for newcomers while supporting complex application development.

Features of Python

Python's features provide flexibility, maintainability, and portability:

- **Easy to Learn:** Minimal keywords and a clear syntax allow rapid learning and implementation.
- **Database Support:** Provides interfaces for integration with major commercial and open-source databases.
- **GUI Development:** Supports graphical user interface development on platforms like Windows, macOS, and Unix-based systems.
- **Scalability:** Supports large-scale application development more effectively than traditional scripting languages.

Additional features include:

- Support for functional and structured programming in addition to OOP.
- Ability to serve as a scripting language or be compiled into bytecode for complex applications.
- High-level dynamic data types and dynamic type checking.
- Automatic memory management and garbage collection.
- Seamless integration with languages and technologies such as C, C++, Java, COM, ActiveX, and CORBA.

Python Libraries Utilized

The project leverages several key Python libraries for data processing, visualization, and machine learning:

- **NumPy:** Provides support for multi-dimensional arrays and numerical operations.
- **Pandas:** Offers data structures like DataFrames for efficient data manipulation and analysis.
- **Matplotlib:** A versatile 2D plotting library for creating publication-quality figures.
- **Scikit-learn:** Implements a wide range of machine learning algorithms for classification, regression, and clustering tasks.

Figure : NumPy, Pandas, Matplotlib, Scikit-learn

SOFTWARE TESTING

Software testing is a systematic process aimed at identifying defects, errors, or weaknesses in a software product. It involves executing the software under controlled conditions to verify that it behaves as expected, meets the specified requirements, and functions reliably under intended operating conditions. Testing is crucial to ensure that individual components, integrated modules, and the complete system perform correctly and do not fail in ways that could impact users or operational integrity. Different types of testing are designed to target specific aspects of the software to guarantee comprehensive quality assurance.

Testing Methodologies

Developing effective testing methodologies begins with creating a structured test plan that outlines the evaluation of general functionality, critical features,

and compatibility across multiple platforms. A rigorous quality control process is employed to verify that the system meets the criteria specified in the software requirements document. The testing framework is designed to systematically uncover defects and ensure that the application operates reliably, efficiently, and according to design specifications.

Unit Testing

Unit testing focuses on verifying the correctness of individual software components or modules in isolation. Test cases are designed to validate internal logic, input handling, and output accuracy. All decision paths and internal workflows are checked to ensure that each component functions as intended before being integrated into the larger system. Unit testing is structural and relies on knowledge of the software's construction. It ensures that every unique process path performs accurately, adhering to documented specifications, with clearly defined inputs and expected outputs.

Test Plan Development

Effective testing begins by breaking the software project into smaller units or modules. Each unit undergoes detailed testing to identify potential defects early in the development process. Unit testing helps isolate and rectify errors before components are integrated into the overall system. By following a structured testing strategy, the software achieves higher reliability, reduced error rates, and improved overall quality.

FUTURE ENHANCEMENTS

The proposed traffic accident detection system can be further improved by adopting advanced deep learning strategies and hybrid model architectures to enhance both accuracy and computational efficiency. One promising approach is the use of spatio-temporal models, such as 3D Convolutional Neural Networks (3D-CNNs) or Vision Transformers (ViTs), which can more effectively capture motion patterns and temporal dependencies across consecutive video frames.

In addition, multi-camera video fusion could be incorporated to address occlusion and blind spot challenges. By integrating views from multiple surveillance angles, the system can achieve more robust and reliable detection in complex traffic environments. Another potential enhancement is automatic accident severity estimation, leveraging visual cues and scene analysis to provide real-time insights for emergency response and traffic management.

Deployment of the system on edge computing devices, coupled with optimization using lightweight versions of YOLO (e.g., YOLOv8n or YOLOv9-lite), would enable real-time processing and scalability for city-wide traffic networks. Furthermore, the development of a user-friendly web-based dashboard for live visualization, event logging, and analytics would enhance accessibility and usability for traffic authorities, facilitating informed decision-making.

Collectively, these improvements would contribute to creating a more intelligent, adaptive, and proactive traffic monitoring solution, capable of supporting large-scale implementations while ensuring timely and accurate accident detection.

Conclusion

In summary, the proposed traffic accident detection system leverages the YOLOv8 deep learning framework to provide an efficient, fully software-driven solution for real-time identification of road accidents. By utilizing YOLOv8's advanced architecture—including transformer-based attention mechanisms, re-parameterized convolutional layers, and decoupled detection heads—the system demonstrates high detection accuracy and rapid inference, even in complex traffic scenarios.

The modular structure of the system enables a seamless workflow from data acquisition and preprocessing to model evaluation and result visualization. This eliminates the need for additional hardware or IoT infrastructure while maintaining reliable performance. Experimental evaluation shows that the system can effectively detect various accident scenarios, such as collisions and vehicle rollovers, under diverse environmental conditions.

Overall, this project highlights the potential of artificial intelligence and computer vision in enhancing road safety, minimizing emergency response times, and laying the foundation for the development of intelligent transportation systems that are both scalable and proactive.

References

1. World Health Organization. *Road traffic deaths*. Available: http://www.who.int/gho/road_safety/mortality/en/
2. J.L. D'Souza, "Road Safety: A Critical Issue Requiring Government Attention," *International Journal of Education and Science Research Review*, vol. 2, no. 2, April 2015.
3. J. Pang, I. Singh, "Accelerometer-Based Real-Time Remote Detection and Monitoring of Hand Motion," *Proceedings of the World Congress on Engineering and Computer Science*, vol. II, Oct. 19–21, 2011, San Francisco, USA.
4. S.J. Weiner, "Feasibility of a 802.11 VANET-Based Car Accident Alert System."
5. U. Halil, T. Javid, A. Nasir, "Automatic Road Accident Detection Techniques: A Brief Survey," *2017 International Symposium on Wireless Systems and Networks (ISWSN)*, 2017.
6. A.B. Faiz, A. Imteaj, M. Chowdhury, "Smart Vehicle Accident Detection and Alarming System Using a Smartphone," *2015*

International Conference on Computer and Information Engineering (ICCIE), 2015.
7. M. Ozbayoglu, G. Kucukayan, E. Dogdu, "A Real-Time Autonomous Highway Accident Detection Model Based on Big Data Processing and Computational Intelligence," *2016 IEEE International Conference on Big Data*, 2016.
8. S. Bhavthankar, H.G. Sayyed, "Wireless System for Vehicle Accident Detection and Reporting Using Accelerometer and GPS," *International Journal of Scientific & Engineering Research*, vol. 6, no. 8, Aug. 2015.
9. M. Chong, A. Abraham, M. Paprzycki, "Traffic Accident Analysis Using Machine Learning Paradigms," *Informatica*, vol. 29, 2005, pp. 89–98.
10. K. O'Shea, R. Nash, "An Introduction to Convolutional Neural Networks," arXiv:1511.08458v2, Dec. 2015.
11. S.J. Pan, Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.
12. N. Dogru, A. Subasi, "Traffic Accident Detection Using Random Forest Classifier," *2018 15th Learning and Technology Conference (L&T)*, 2018.
13. J. Jayadeva, R. Khemchandani, S. Chandra, "Twin Support Vector Machines for Pattern Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, pp. 905–910, 2007.
14. S. Ding, J. Yu, B. Qi, H. Huang, "An Overview on Twin Support Vector Machines," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 245–252, 2012.
15. M. Tanveer, "Application of Smoothing Techniques for Linear Programming Twin Support Vector Machines," *Knowledge and Information Systems*, vol. 45, no. 1, pp. 191–214, 2015.
16. B. Richhariya, M. Tanveer, "A Robust Fuzzy Least Squares Twin Support Vector Machine for Class Imbalance Learning," *Applied Soft Computing*, vol. 71, pp. 418–432, 2018.
17. D. Li, Y. Tian, "Twin Support Vector Machine in Linear Programs," *Procedia Computer Science*, vol. 29, pp. 1770–1778, 2014.
18. P. Kaliuga Lakshmi, C. Thangamani, "An Efficient Vehicle Accident Detection Using Sensor Technology," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 5, no. 3, Mar. 2016.
19. P. Anudeep, K.H. Babu, "Wireless Reporting System for Accident Detection at Higher Speeds," *Int. Journal of Engineering Research and Applications*.
20. M. Dbrowski, T. Michalik, "Effectiveness of Transfer Learning Methods for Image Classification," *Federated Conference on Computer Science and Information Systems*, pp. 39.
21. X. Xia, C. Xu, B. Nan, "Inception-v3 for Flower Classification," *2017 2nd International Conference on Image, Vision and Computing*, 2017.
22. S.A. Prajapati, R. Nagaraj, S. Mitra, "Classification of Dental Diseases Using CNN and Transfer Learning," *2017 5th International Symposium on Computational and Business Intelligence (ISCBI)*, 2017.