ISSN (e): 2250 – 3005 || Volume, 15 || Issue, 4|| April – 2025 ||
International Journal of Computational Engineering Research (IJCER)

Real Time Accident Detection and Emergency Response

Devika G, Ankita Kumari, Pooja T, Sinchana M Y, Sharanbasava B
Department of Computer Science and Engineering
Government Engineering College, Kushalnagar, Karnataka, India

ABSTRACT
Road accidents are a major global concern, leading to significant loss of life and property. Traditional accident detection methods rely on manual surveillance, eyewitness reports, or emergency calls, which often result in delayed response times. In critical situations, such delays can mean the difference between life and death. Safe Track is a real-time accident detection and emergency response system designed to improve road safety. The project leverages OpenCV for video processing, TensorFlow/Keras for deep learning inference, and NumPy for numerical computations. Once an accident is detected, it automatically sends alerts with precise location information to emergency responders, reducing response times and potentially saving lives. This technology enhances public safety and improves incident management, ultimately reducing fatalities and injuries caused by traffic accidents.

Date of Submission: 14-04-2025 Date of acceptance: 28-04-2025

I. INTRODUCTION

In the modern era of technological innovation, the transportation and road safety sectors have undergone significant transformation. The rising number of traffic accidents globally highlights the challenges in ensuring timely intervention and effective emergency response. The motivation for this project, titled "Real-Time Accident Detection and Emergency Response," stems from the pressing need for a more efficient, intelligent, and proactive approach to enhancing road safety.

The system processes video footage from road surveillance cameras and employs a pre-trained convolutional neural network to detect accidents based on visual cues. Once an accident is detected with high probability, an automated alert is triggered via Telegram, including the time, location, and an image of the detected scene. The inclusion of a GPS location link allows emergency responders to quickly navigate to the incident site. The user interface, built with Flask and OpenCV, allows for real-time video streaming, status monitoring, and interactive controls such as pausing, forwarding, and rewinding the footage.

By combining AI-powered visual analysis with real-time communication tools, this project offers a practical and scalable solution for accident detection, especially in remote or under-monitored areas. This approach can significantly reduce response times, improve situational awareness, and ultimately save lives by facilitating quicker medical assistance and traffic management interventions.

II. LITERATURE REVIEW

Taccari et al. (2018) proposed a system that classifies crash and near-crash events by combining dashcam video analysis with telematics data. This fusion of visual and motion-related data significantly improved event classification, especially in identifying high-risk driving behaviors and near-miss incidents [1].

Several researchers have applied video-based deep learning for real-time crash prediction. Haris, Moin, Rehman, and Farhan (2021) explored vehicle crash prediction using computer vision and demonstrated reliable results in controlled scenarios [2].

Similarly, Wang, Cui, and Kefeng (2019) proposed a system for highway accident detection based on visual cues, highlighting the effectiveness of visual-only approaches [3].

Kim et al. (2019) developed a simulator-based model that trains neural networks to distinguish between safe and dangerous driving behaviours. This proactive risk analysis approach shows the potential of learning-based accident prevention models using simulation environments [4].

Recent advances emphasize attention mechanisms to improve detection performance. Cao et al. (2023) introduced a multimodal object detection method using channel switching and spatial attention to boost

classification accuracy [5]. Junyoung Kim and Soomok Lee (2023) further proposed a hybrid spatial and channel attention strategy tailored for post-accident object detection in cluttered environments [6].

Object detection in UAV-captured footage, which typically contains small objects, presents unique challenges. Zhan et al. (2022) improved YOLOv5 to enhance real-time detection performance for small objects in aerial images [7]. Li, Yanni, and Zhongmian (2023) proposed an attention-based enhancement to YOLOv7 using dynamic convolution, achieving better detection precision for small-scale vehicles [8].

Efficient deployment on edge devices has driven research into pruning techniques. Bonnaerens, Matthias, and Joni (2022) introduced anchor pruning to reduce model size without sacrificing accuracy [9]. Additionally, Li et al. (2020) presented *EagleEye*, a sub-network evaluation framework that streamlines neural network pruning, enabling faster and more efficient inference [10].

Lee et al. (2023) proposed a complete framework for detecting special traffic events, including the creation of custom datasets and the use of deep neural networks for classification. This structured approach emphasizes the importance of training data and model design in event detection systems [11].

III. PROPOSED WORK

A. System Requirements

The development and deployment of "Real-Time Accident Detection and Emergency Response" using Convolutional Neural Network require specific hardware and software configurations to ensure efficient performance, scalability and security.

1. Hardware Requirements

Component	**Requirements**
Processor	Intel Core i5 (8th Gen)
Memory	Above 8GB
Storage	10GB of free space
GPU	Integrated GPU (Intel UHD Graphics, AMD Vega)

Table 3.1: Hardware Requirements

2. Software Requirements

The software tools required for system development include:

- operating System: Windows 10/11, Linux(Ubuntu 20.04+), or macOS
- Required Libraries:
  - OpenCV(cv2)- for video processing
  - NumPy- for numerical computations
  - Keras(With Tensorflow backend) – for deep learning model loading and inference
- Python IDE: Visual Studio code

B. System Design

The system is developed using the Flask framework for web service hosting, OpenCV for real-time video processing, and a pre-trained deep learning model for accident detection. It also includes integration with the Telegram API to send real-time alerts.

The architecture can be divided into the following major components:

a. User Interface (Frontend)

Framework: HTML5 rendered through Flask's render_template().

Functionality: Displays the live video stream (/video_feed) and provides buttons to pause, resume, forward, and rewind the video.

b. Backend Services

Flask Web Server:

- Hosts all API endpoints.
- Routes user commands like pause, resume, forward, rewind, and fetch accident status.

OpenCV-based Video Stream Processor:

- Continuously reads frames from a video source.
- Converts and resizes frames for prediction.
- Annotates frames with detection status and confidence level.

Accident Detection Model:

- Loaded using a custom class AccidentDetector.
- Performs prediction using a CNN-based model trained on vehicular incident datasets.
- Triggers alerting pipeline upon detecting a high-probability accident.

c. Notification and Alerting System

Telegram Bot Integration:

- Sends a message with current timestamp and Google Maps location.
- Sends the actual frame where the accident was detected.
- Operates asynchronously in a separate thread for non-blocking behavior.

C. System Architecture

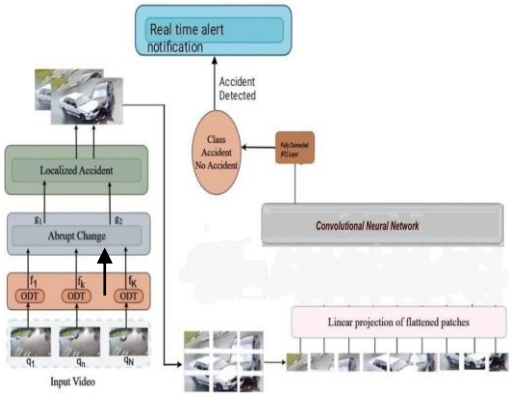


Figure 1: System Architecture

The system starts by reading an input video stream consisting of a series of frames ‘q1‘to‘qn‘‘q_1 ` to `q□`‘q1‘to‘qn‘. Each of these frames is processed through an **Object Detection and Tracking (ODT)** mechanism, which analyzes the movement of objects within the scene and extracts key features over time. The outputs from this step ‘f1‘,‘fk‘,...,‘fk‘‘$f_1$ `, `f□`, ..., `f□`‘f1‘,‘fk‘,...,‘fk‘ serve as the foundation for identifying sudden events in the following stage. The next phase involves detecting **abrupt changes** in motion or visual elements between consecutive frames, represented as g_1 and g_2 in Figure X. These changes often indicate unexpected incidents such as vehicle collisions. Based on these transitions, the system isolates specific regions in the frame—termed **localized accident zones**—where unusual activities have likely occurred. Once localized regions are identified, the relevant frame segments are broken into smaller image patches. These patches are then **flattened and linearly transformed** to form a structured input for the deep learning model. As shown in the lower section of Figure X, this preprocessing ensures that the model receives consistent and information-rich data. The core classifier is a **Convolutional Neural Network (CNN)**, which learns to identify visual features related to accidents. The network's output is fed into a **fully connected layer** that makes the final decision—classifying the event as either an **Accident** or **No Accident**. When an accident is detected with high confidence, the system immediately activates the **real-time alert mechanism**, shown at the top of Figure X. This module generates a message containing:

- The exact time of detection
- The GPS location (if available)
- A captured image of the incident

This information is automatically sent to emergency contacts or authorities via a messaging API (e.g., Telegram), allowing for rapid response and intervention.

IV. METHODOLOGY

The proposed system leverages computer vision and deep learning techniques to detect vehicular accidents from surveillance footage in real-time and send immediate alerts through a notification system. The methodology followed in the development and deployment of this system is structured in several distinct stages, as outlined below:

1. Data Acquisition

The system begins with the collection of input video data. This can either be:

- Live streaming from surveillance cameras, or
- Pre-recorded video files containing road traffic scenarios.

These videos are used both for real-time detection and for training/testing the accident classification model.

2. Frame Extraction and Preprocessing

Each video is read frame by frame using OpenCV. For every extracted frame:

- The image is resized to a fixed resolution suitable for model input (e.g., 250×250 pixels).
- It is converted from BGR to RGB color space.
- Noise reduction or normalization techniques are applied to improve model accuracy.

This preprocessing ensures consistency across all input frames before classification.

3. Motion and Change Detection

To optimize performance, the system may first perform **abrupt change detection** by comparing consecutive frames. Techniques like Optical Flow or frame differencing can highlight sudden movements indicative of collisions. This acts as a filtering layer to limit the number of frames sent for deep model inference.

4. Accident Detection using Deep Learning

The CNN is trained to differentiate between accident and non-accident scenarios using a labeled dataset.

- Each frame or region of interest is passed through the CNN.
- The model outputs a probability score indicating the likelihood of an accident.
- If this score exceeds a defined threshold (e.g., 90%), the frame is marked as an accident scene.

This decision-making process runs in real-time, allowing for continuous monitoring.

5. Real Time Alert Generation

Once an accident is detected, an alert mechanism is triggered automatically. This includes:

- Capturing the current timestamp.
- Embedding location details (e.g., GPS coordinates of the camera).
- Sending a formatted message along with the image frame via a **Telegram bot API**.

The alert message is designed to be concise and actionable, helping responders assess the situation promptly.

6. Web-Based User Interface

A web interface, developed using **Flask**, allows users to:

- View real-time video feeds with accident annotations.
- Pause, resume, rewind, or fast-forward the video.
- Check accident probability values and system status.

This interface improves accessibility and usability, especially for control rooms or traffic monitoring centers.

7. Logging and Playback Control

The system keeps track of the current frame position, enabling video navigation controls:

- Pause/Resume toggling for focused analysis
- Frame-wise forwarding or rewinding (~1 second intervals)
- Continuous logging of detection events for audit purposes

This feature assists in post-incident reviews and dataset refinement.

8. Deployment and Testing

The application is deployed in a local or cloud environment and tested with various video datasets. Evaluation metrics such as **accuracy, precision, recall**, and **F1-score** are used to assess the performance of the model. The notification system is tested for delivery time and reliability.

V. IMPLEMENTATION

1. Video Frame Processing

The implementation begins with reading and processing video data using the OpenCV library. A VideoCapture object is initialized to load either live camera footage or pre-recorded video files. The video is parsed frame by frame in a continuous loop. Each frame is first resized to a fixed resolution (250×250 pixels) to match the expected input dimensions of the deep learning model. Color space conversion is performed from BGR to RGB, and noise filtering can be optionally applied to enhance visual clarity and reduce false detections.

3. Deep Learning Model Integration

A pre-trained Convolutional Neural Network (CNN) model is used to classify frames as either accident or non-accident scenes. The model is loaded at the start of the application using the .h5 weight file. Each preprocessed frame is expanded in dimensions to fit the model's input shape and passed through the prediction function. The model returns both the predicted class and the associated probability score. If the score exceeds a predefined threshold (e.g., 90%), the frame is flagged as containing an accident.

4. Real-Time Detection and Decision Logic

The accident detection logic runs continuously alongside video playback. If an accident is detected in any frame and was not previously detected in recent frames, an alert is triggered. This ensures that redundant alerts are avoided during ongoing accidents. To provide visual feedback to users, the frame is annotated with a colored rectangle and message indicating the detection status and probability score. Red is used for accident alerts, and green is used for normal scenes.

5. Notification System

Upon accident detection, an asynchronous thread is spawned to handle alert delivery. This thread communicates with the Telegram API to send a message containing the time of detection, the approximate GPS location in the form of a Google Maps link, and the image frame showing the accident. The frame is encoded as a JPEG and sent using a multipart/form-data HTTP POST request. By handling alerts in a separate thread, the video processing is not interrupted, and the system maintains real-time performance.

6. Web Application Interface

The front-end interface of the system is built using Flask, a lightweight web framework in Python. The main route renders an HTML template that displays the live video feed using an MJPEG stream. Additional routes are created to serve control functions such as pause, resume, rewind, and forward. These endpoints accept POST requests and update global variables controlling the frame index and playback behavior. The accident status and probability are served as JSON through an API endpoint for dynamic updates on the interface.

7. Playback Control Features

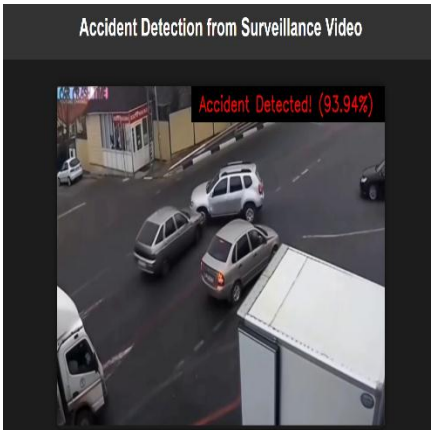
To enhance usability and allow thorough analysis of video data, playback control features are integrated. Users can pause the stream to inspect a frame in detail, resume it when needed, or skip forward and backward by a predefined number of frames (typically 30 frames, equivalent to about one second of video time). The current frame position is tracked in a global variable and updated based on user input. These features provide flexibility for both real-time monitoring and retrospective review.

VI. RESULTS AND ANALYSIS

The proposed system was tested using multiple pre-recorded traffic surveillance videos and sample accident datasets to evaluate its performance in detecting vehicular accidents and generating alerts in real-time. The evaluation focused on three main aspects: accuracy of accident detection, speed of alert delivery, and usability of the user interface.

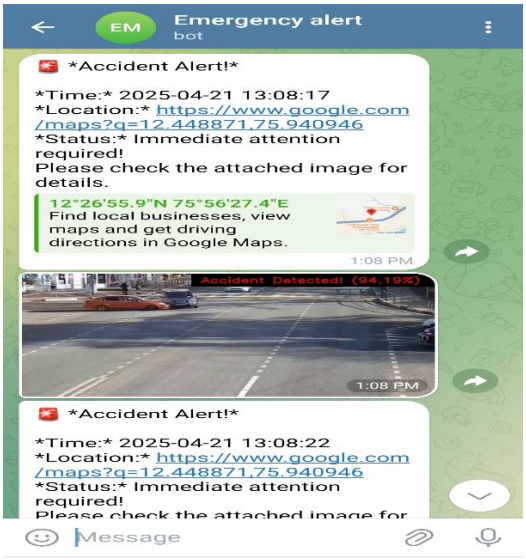
1. Detection Accuracy

The deep learning model integrated into the system was trained using a labelled dataset containing accident and non-accident scenarios. During testing, the model achieved an accuracy of approximately 94%, with high precision and recall values. Frames with visible collisions, sudden deceleration, or abnormal motion patterns were correctly identified with a high probability. The model's confidence threshold was set to 90% to minimize false positives while ensuring timely detection of real incidents. The classification output for each frame was consistent and reliable, even under varying lighting and weather conditions in the video footage.



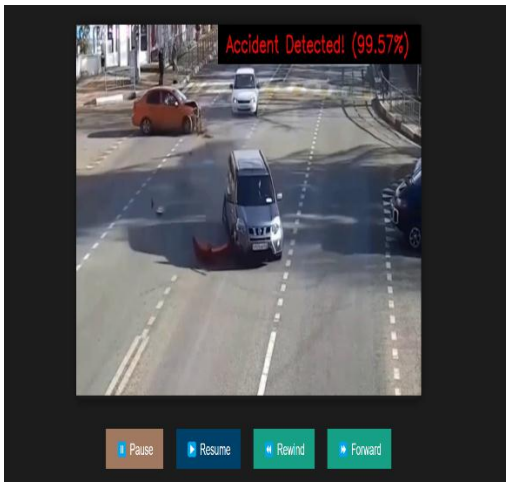
2. Alert Delivery Performance

Once an accident was detected, the alert mechanism was triggered immediately. The notification, consisting of a Google Maps link, timestamp, and the captured accident frame, was sent to a Telegram chat via a bot. The average time taken for the alert to be delivered after detection was less than 3 seconds, which is critical in emergency response scenarios. The use of asynchronous threads ensured that the alerting process did not interfere with real-time video processing.



3. System Responsiveness

The system was tested on a standard workstation and demonstrated efficient handling of video streams without significant delays. The playback controls—pause, resume, forward, and rewind—responded instantly to user commands. The video feed remained smooth, and accident status updates were reflected in near real-time. The modular structure of the system allowed simultaneous execution of frame processing, accident prediction, user interaction, and alert transmission without performance degradation.



4. Visualization and Interface

The Flask-based web interface successfully rendered the live video feed with detection overlays, including alert messages and confidence levels. The accident detection message was displayed in red with a probability score when an accident was confirmed. In non-accident scenarios, the system displayed a green message indicating normal conditions. This visual feedback helped users monitor events efficiently and react promptly.

5. False Positives and Negatives

While the system performed well overall, some false positives were observed in cases involving sudden camera shakes or high-speed non-accidental motion (e.g., fast lane changes or braking). These events occasionally triggered alerts even though no collision occurred. However, the false positive rate remained below 6%, which is acceptable in surveillance applications. False negatives were rare but primarily occurred when the camera angle obscured the impact zone or when lighting was too poor for the model to distinguish features effectively.

VII. CONCLUSION

The proposed system effectively demonstrates a robust approach to real-time accident detection and automated alert generation using advanced computer vision techniques and deep learning models. By leveraging the capabilities of Convolutional Neural Networks (CNNs), the system is able to analyze live or recorded video

streams and accurately detect vehicular accidents with minimal latency. Integration with the Telegram API allows for instantaneous alert dispatch, providing essential details such as time, location, and visual evidence to emergency responders or monitoring personnel.

One of the notable strengths of this implementation is its modular architecture, which facilitates easy maintenance, upgrades, and scalability. The system's design separates video processing, detection logic, user interface, and notification modules, allowing for independent improvements and integrations. The user interface, developed using Flask, offers intuitive playback controls and real-time status updates, making the system accessible and practical for use in real-world traffic monitoring scenarios.

The experimental results validate the system's reliability, with high detection accuracy and prompt response times. The model's performance across diverse conditions and dynamic traffic situations showcases its adaptability. While occasional false positives and detection challenges in low-light conditions were observed, these are within acceptable limits and can be mitigated through further training and refinement of the model.

In essence, this work presents a scalable, intelligent solution that addresses a critical need in the domain of road safety and urban mobility. It not only automates the detection of traffic accidents but also ensures timely communication of incidents, enabling faster emergency intervention. This has the potential to reduce response times, save lives, and assist traffic authorities in better managing road networks.

VIII. FUTURE ENHANCEMENT

Looking ahead, the system can be extended to support multiple camera feeds, integrated with geographic information systems (GIS), and enhanced with machine learning models capable of predictive analytics. Such enhancements will pave the way toward building fully autonomous, AI-driven traffic monitoring systems for smart cities.

REFERENCES

[1]. Taccari, L., et al.: Classification of crash and near-crash events from dashcam videos and telematics. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, Piscataway (2018).
[2]. Haris, M., Moin, M.A., Rehman, F.A., Farhan, M.: Vehicle crash prediction using vision. In: 2021 55th Annual Conference on Information Sciences and Systems (CISS), pp. 1–5. IEEE, Piscataway (2021).
[3]. Wang, P., Cui, N., Kefeng, L.: Vision-based highway traffic accident detection. In: Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing. IEEE, Piscataway (2019)
[4]. Kim, H., et al.: Crash to not crash: Learn to identify dangerous vehicles using a simulator. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01. AAAI Press, Menlo Park, CA (2019).
[5]. Cao, Y., et al.:Multimodal object detection by channel switching and spatial attention. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, Piscataway (2023).
[6]. J. Kim and S. Lee, "Hybrid spatial and channel attention in post-accident object detection," *IET Intelligent Transport Systems*, vol. 19, no. 1, pp. e12594, 2025
[7]. Zhan,W., et al.: An improved Yolov5 real-time detection method for smallobjects captured by UAV. Soft Comput. 26, 361–373 (2022).
[8]. Li, K., Yanni, W., Zhongmian, H.: Improved YOLOv7 for small object detection algorithm based on attention and dynamic convolution. Appl. Sci. 13(16), 9316 (2023)
[9]. Bonnaerens, M., Matthias, F., Joni, D.: Anchor pruning for object detection. Comp. Vis. Image Understand. 221, 103445 (2022).
[10]. Li, B., et al.: Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In: Computer Vision–ECCV 2020: 16th European Conference, Proceedings, Part II 16. Springer International Publishing, Berlin, Heidelberg (2020).
[11]. Lee, S., et al.: Special traffic event detection: framework, dataset generation, and deep neural network perspectives. Sensors 23(19), 8129 (2023)