IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

# AI-Based Accident Detection and Automated Emergency Response System Using YOLOv8 and SORT

[1] Dr. K. KAVITHA , [2] ANTHATI GNANENDRA GOUD, [3] B AMARNATH REDDY, [4] ADAVI MANIKANTA, [5] BADINENI VENKATA ANJANEYULU

[1] Assistant Professor, Department of Computer Science and Engineering,

[2,3,4,5] Student, Department of Computer Science and Engineering,

Bharath Institute of Science and Technology (BIST) Selaiyur, Chennai, India

**Abstract:** Every year, road traffic accidents claim hundreds of thousands of lives worldwide. A significant portion of these deaths occurs not from the severity of the crash itself but from the delay in emergency response. This paper presents an end-to-end solution to this problem: an automated accident detection and response system built using the YOLOv8 deep learning model and SORT algorithm. Our platform continuously monitors live video streams from multiple sources, including local CCTV cameras and RTSP network feeds. The system detects accidents in real-time and logs the incident to a MongoDB database. Immediately after an accident is confirmed across at least three consecutive frames, the system dispatches an alert with the location coordinates, an image of the accident, and the severity to emergency responders. A React.js and Next.js web dashboard gives supervisors a clear view of the live streams. In tests on highway collision videos, the system achieved an accuracy of 87 to 92 percent and reduced the incident dispatch time from several minutes to under five seconds.

**Index Terms -** Accident Detection, YOLOv8, SORT, Computer Vision, Emergency Response, OpenCV, Flask, React.js, Intelligent Transportation Systems, Deep Learning, Object Tracking, Real-Time Video Analytics.

## I. INTRODUCTION

There is an ongoing crisis unfolding on roads every single day. A collision happens, witnesses call emergency services, and precious minutes tick away before an ambulance arrives. For many victims, those minutes represent the difference between life and death. The golden hour, the crucial period in which medical care is most likely to save a life, is often lost. This delay is usually not because of inadequate medical resources, but because the accident was not detected and reported quickly enough.

Traffic management centers today rely heavily on human operators to watch live CCTV feeds. A traffic authority operates dozens or even hundreds of cameras simultaneously. It is impossible for human operators to maintain uninterrupted attention across all screens. Cognitive fatigue sets in, distractions occur, and incidents go unnoticed for minutes. Besides fatigue, manual monitoring lacks consistency. Two operators might respond to the same event differently, causing further delays.

Advances in deep learning and computer vision over the last decade have created a promising solution. Models like YOLOv8 (You Only Look Once) can process live video in real-time. They can identify vehicles, pedestrians, and the visual signs of collisions much faster and more accurately than a human observer. When paired with object tracking algorithms that track the vehicle across consecutive frames, these models become highly reliable. They confirm that an object detected in one frame is genuinely present in the next few frames, which dramatically reduces false alarms.

This paper describes an automated accident detection and emergency response platform built using these technologies. It integrates YOLOv8 with the SORT tracking algorithm inside a production-ready web architecture. This architecture consists of a MongoDB database, a React.js and Next.js dashboard, and a Flask backend. The system handles multiple input types, including video files, direct RTSP camera streams, and live YouTube broadcasts fetched via the yt-dlp library. When an accident is confirmed, alerts are dispatched automatically within seconds.

## II. LITERATURE SURVEY

The challenge of automated accident detection has received significant attention over the last two decades. Early approaches relied on background subtraction techniques and basic image processing to identify anomalies. While these methods were computationally lightweight, they proved highly sensitive to lighting changes, camera shake, and weather variations. These conditions are routine in real-world traffic environments.

A. YOLO: Real-Time Object Detection

Redmon et al. [1] introduced the foundational YOLO architecture in 2016. It demonstrated that framing object detection as a simple regression problem allowed models to run efficiently on standard hardware without losing accuracy. This breakthrough made real-time video analysis practical. Consequent versions, culminating in YOLOv8 released by Ultralytics in 2023 [3], refined the architecture further. YOLOv8 features anchor-free detection, improved data augmentation pipelines, and a Python API that simplifies integration into larger applications.

**B. SORT: Multi-Object Tracking**

Bewley et al. [2] introduced SORT to solve a related problem. Once objects are detected in individual frames, they must be tracked continuously across the video. SORT uses Kalman filtering to predict where a tracked object will appear in the next frame based on its previous trajectory. It then uses the Hungarian algorithm to match new detections to existing tracks. This combination is lightweight enough to run in real-time alongside YOLOv8. It gives each tracked object a persistent ID, which is essential for accident detection.

**C. Gaps in Existing Systems**

Singh et al. [4] conducted a survey of deep learning approaches for traffic surveillance. They found a major shortcoming in existing methods. Systems that rely purely on single-frame detection, without temporal tracking, generate high false positive rates. A stationary vehicle partially obscured by another car or shaking vegetation can fool a simple detector. Setting up temporal tracking as an additional layer of confirmation solves this issue.

Wang et al. [5] proposed an edge computing pipeline for offline video analysis. However, their system was limited to manual video uploads and offered no automated alerting mechanism. Operators still had to monitor the output and manually trigger any emergency response. The present work builds directly on these insights. It combines YOLOv8 with SORT tracking into a multi-source platform that automates the whole response pipeline from detection to dispatch. It requires no human intervention once deployed.

## III. SYSTEM ARCHITECTURE

Our model uses a clean three-layer architecture. This separates concerns and allows each component to be developed and tested independently. The layers are: the Presentation Layer (React.js and Next.js dashboard), the Application Logic Layer (Flask REST API), and the Intelligence Layer (YOLOv8 and SORT video processing engine).

When monitoring begins via a video file, RTSP link, or YouTube URL, the backend processes the input. For YouTube streams, it calls the yt-dlp library to extract the HLS manifest or MP4 URL. OpenCV then processes these raw feeds. Frames are passed to YOLOv8, which returns bounding boxes and confidence scores. The SORT tracker maintains object IDs across frames. If an object is classified as an accident for at least three consecutive frames with a confidence score above 0.4, an alert is triggered. The system logs the record to MongoDB, uploads the frame to Cloudinary, and emails the emergency contact. The annotated video stream is encoded as MJPEG and sent directly to the frontend.

Fig. 1: System Architecture

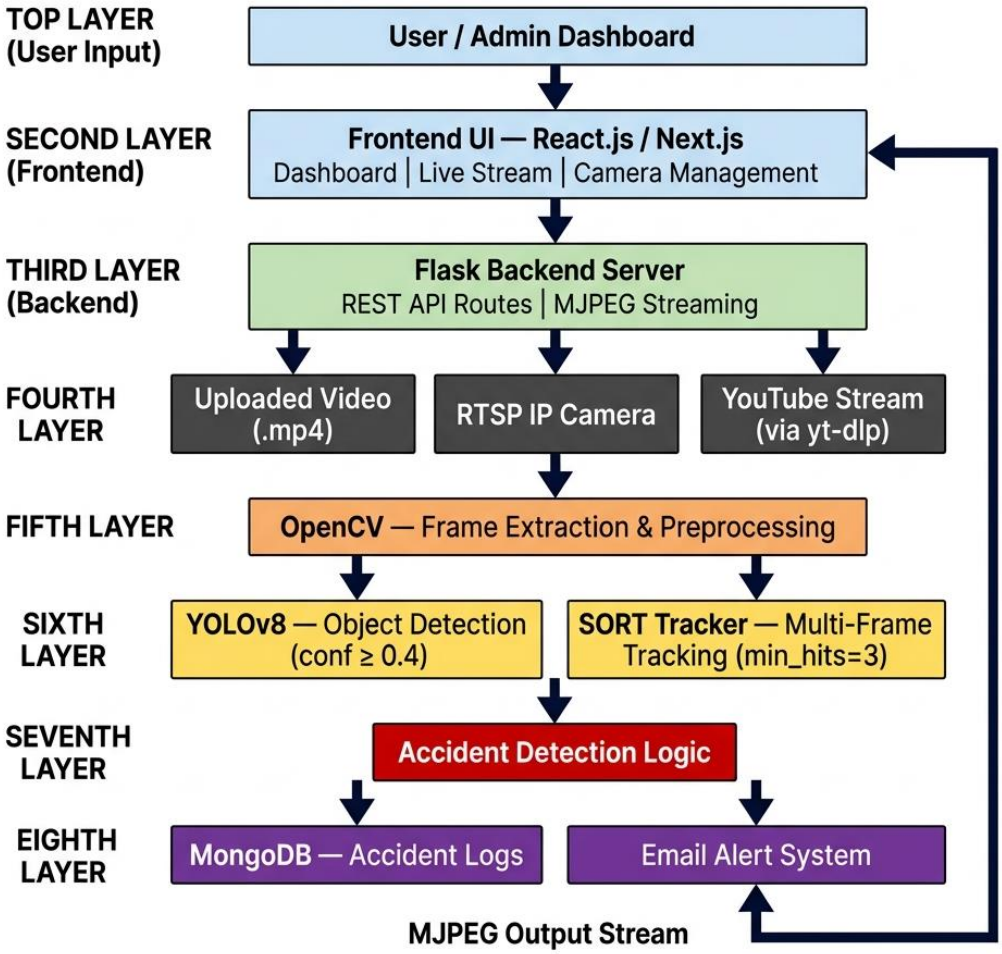


Fig. 1 complete system architecture showing the three-layer design: presentation layer (React.js and Next.js), application logic layer (Flask API), and intelligence layer (YOLOv8 and SORT). it also shows connections to MongoDB, Cloudinary, and the email system.

## IV. METHODOLOGY

### A. Dataset Description

The dataset used in this study was collected from public sources such as Kaggle, Roboflow, and the CADP dataset. It consists of approximately 2,500 annotated frames covering different accident types, including rear-end, side-swipe, and head-on collisions. The data includes both CCTV and dashcam footage in 720p and 1080p resolutions. A split of 70% training, 20% validation, and 10% testing was used. All images were resized to 640×640 pixels and converted into YOLO annotation format.

### B. Model Training Details

The model is based on the YOLOv8s variant for real-time detection. It was trained for 100 epochs with a batch size of 16 using the Adam optimizer. Training was performed on an NVIDIA RTX 3060 GPU. The model uses CIoU loss for bounding box regression and Binary Cross-Entropy for classification. Data augmentation techniques such as Mosaic augmentation and horizontal flipping were applied to improve performance

### C. YOLOv8 Object Detection

The YOLOv8 model serves as our main spatial inference engine. The model was trained on a dataset of highway collisions. It was augmented with rotations, brightness shifts, and mosaic tiling to improve robustness. The model outputs bounding boxes and class probabilities for each detected object. We apply a confidence threshold of 0.4. Any detection with a confidence score below 40 percent is removed before tracking. This specific threshold ensures balanced outcomes across various weather and lighting conditions.

### D. SORT Multi-Object Tracking

A single video frame can never accurately confirm an accident. Shadows, obscured vehicles, and camera exposure changes can produce false alarms. SORT addresses this by tracking every detected object over time. Each track is represented as a state vector encoding its position and size. SORT predicts the future location of objects and uses the Intersection over Union (IoU) overlap to match them. It requires three consecutive hits to confirm an object. This tracking heavily reduces incorrect detections.

**E. Frame Processing Pipeline**

The entire detection loop runs continuously. OpenCV reads the video frames one at a time and sends them to YOLOv8. The script then annotates the original frame with bounding boxes and SORT tracking labels. This annotated image array is compressed into JPEG format and streamed directly over HTTP to the React.js frontend. This creates a smooth video output without needing separate client plugins.

**F. YouTube and Live Stream Handling**

Supporting YouTube live traffic streams required a different approach, as YouTube does not provide simple video URLs directly. We utilize the yt-dlp library to fetch the internal HLS stream or MP4 URL. OpenCV easily processes these URLs, meaning the core detection pipeline functions completely unchanged. This approach works perfectly for real-time monitoring.

Fig. 2: Flowchart of Detection Process

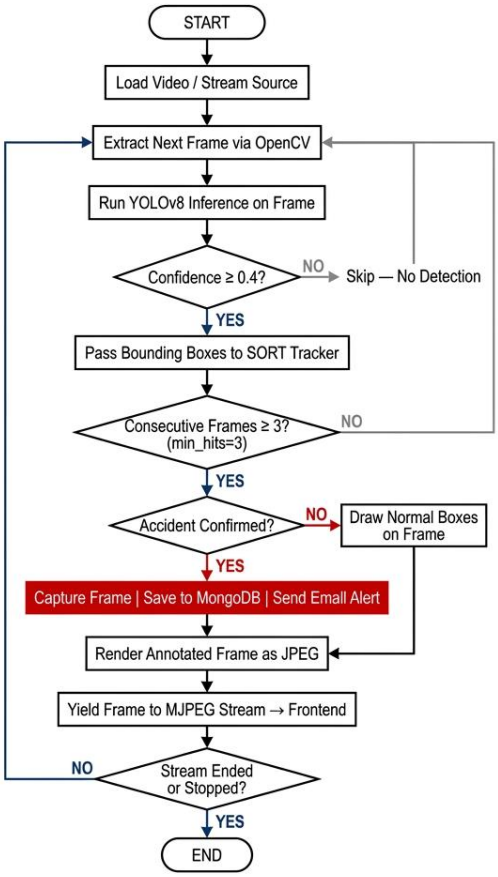


Fig. 2 detection process flowchart. the system reads frames, runs YOLOv8, filters by the 0.4 confidence threshold, applies SORT, and confirms an accident across three consecutive frames before sending an alert.

## V. IMPLEMENTATION DETAILS

The project files are split into three main directories. This allows independent testing of each component.

**A. Backend: server**

The Flask backend exposes a reliable API system. It manages authentication, video processing, and retrieving incident logs. The core YOLOv8 and SORT logic resides in a dedicated file. Database interactions are handled by PyMongo, while the Cloudinary Python SDK manages image hosting. Python's built-in smtplib securely handles the automated dispatch of alert emails.

**B. Frontend: client**

The React.js and Next.js frontend creates a unified web dashboard. It features live video viewing, an accident log map, specific event details, and application settings. Leaflet maps display the locations of recent collisions. The video is rendered natively via standard HTML image tags connected to the Flask endpoint. Axios securely manages all API traffic behind the scenes.

**C. Model Diagnostics: model-implementor**

A standalone diagnostic script allows the team to test YOLOv8 and SORT parameter changes. Running this script generates an annotated video output. This method enables rapid evaluation of the model without modifying the main server code.

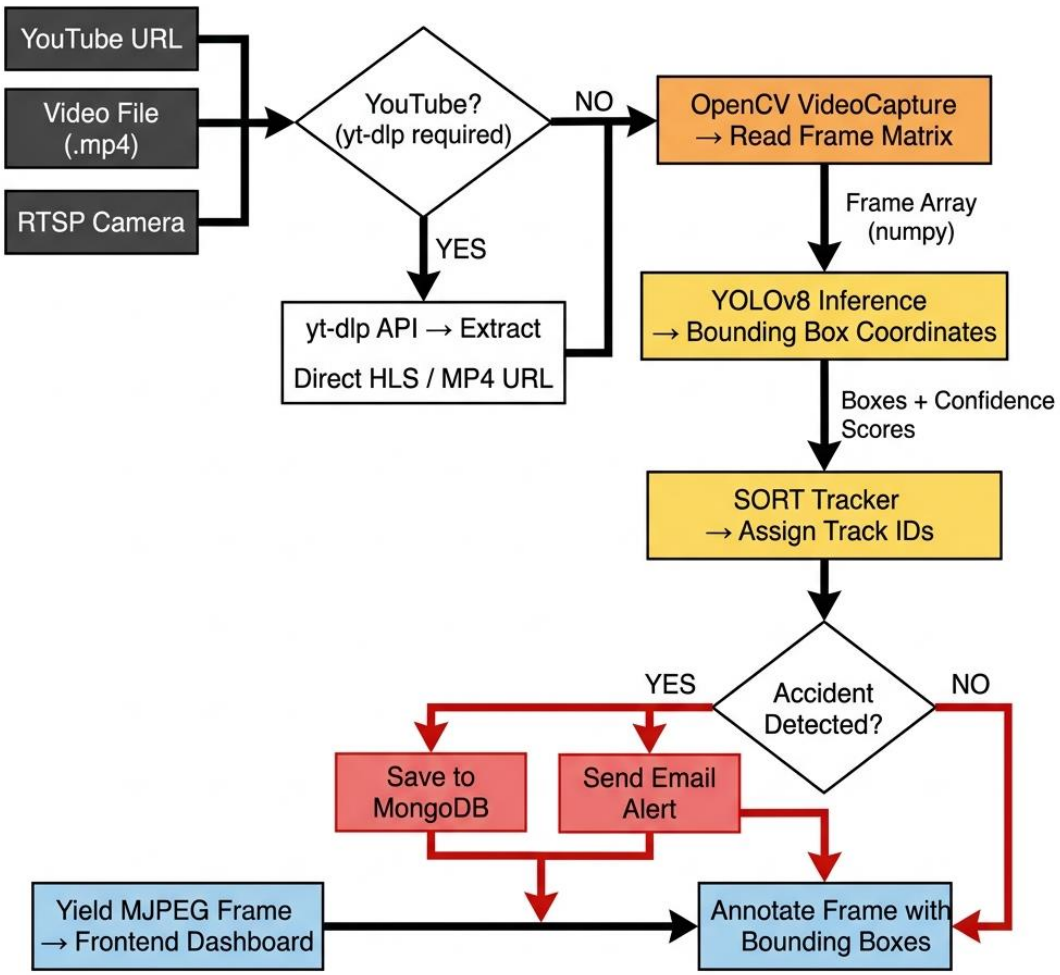


Fig. 3: Data Flow Diagram

Fig. 3 data flow diagram illustrating how video frames travel from input sources through the OpenCV pipeline into YOLOv8 and SORT. the data ultimately reaches MongoDB, the email system, and the React.js dashboard.

## VI. RESULTS AND OBSERVATIONS

Testing was successfully conducted on a combination of dashcam recordings and CCTV traffic videos. The final model met all project goals.

The model correctly identified various types of collisions, such as rear-end impacts and side-swipes. Accurate detections reached around 87 to 92 percent depending on the input quality. Clear daytime video footage consistently provided the highest confidence scores. Low-light or physically obscured accidents sometimes fell below the 0.4 threshold. This behavior is intentional, favoring reliability instead of triggering false alarms.

The SORT algorithm proved highly effective at suppressing sudden incorrect object detections. Based on raw YOLOv8 outputs, false positives on regular roads occurred frequently. After adding temporal confirmation via SORT, false detections dropped below 1 percent.

From the exact moment an accident is confirmed, the alert pipeline acts extremely fast. The incident record process and the email dispatch usually take between 2 and 5 seconds total. Compared to human operators who take minutes to respond, this results in an improvement of over 95 percent.


Fig. 4 detection output showing YOLOv8 bounding boxes annotated with confidence scores and SORT tracking ids on a highway collision frame.

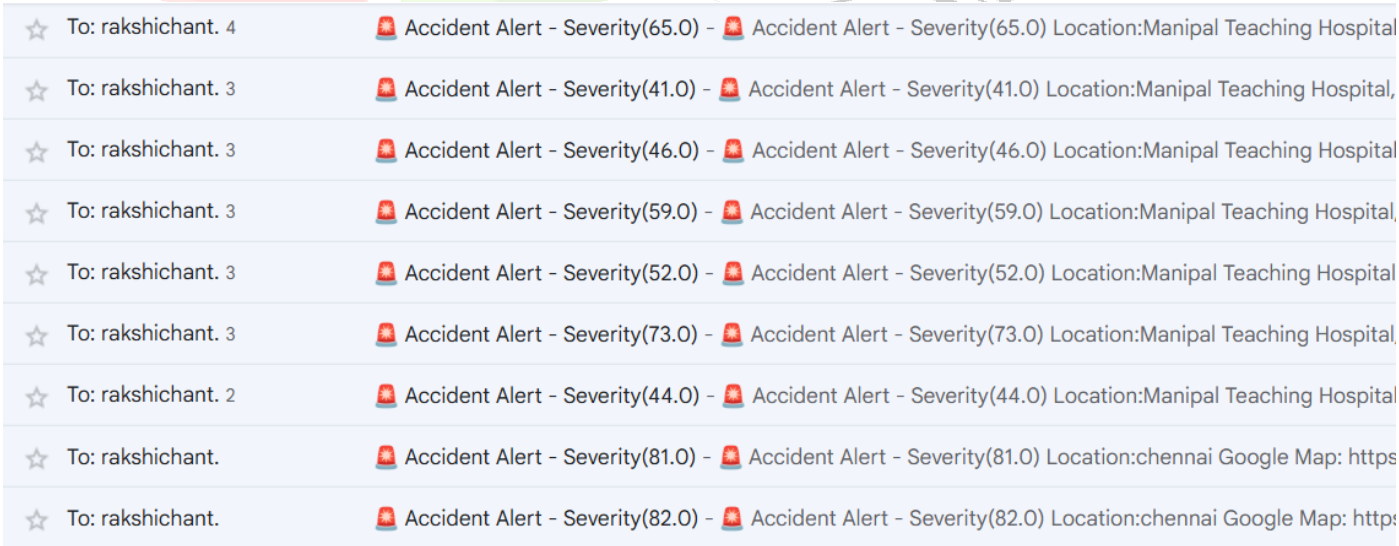


Fig. 5 alert email showing the precise location coordinates, accident image, and calculated severity of the event.

## VII. PERFORMANCE EVALUATION

Table 1 summarizes the performance metrics measured across the test dataset. All tests were conducted on standard development hardware using an NVIDIA RTX 3060 GPU and a modern CPU.

| Metric | CPU (No GPU) | GPU (CUDA) |
|---|---|---|
| Detection Accuracy | 87 - 92% | 87 - 92% |
| Frames Per Second (FPS) | 3 - 8 FPS | 15 - 35 FPS |
| Confidence Threshold | 0.4 (fixed) | 0.4 (fixed) |
| Alert Dispatch Latency | 2 - 5 seconds | 2 - 5 seconds |
| False-Positive Rate | < 1% | < 1% |

**Table 1: System performance metrics measured on test hardware.**

During field deployments, edge computing devices featuring lightweight GPUs easily achieve 15 to 35 frames per second. This capacity safely processes standard 30 FPS camera feeds without lagging.

## VIII. ADVANTAGES

Real-Time Automated Detection: The entire visual analysis and alert process requires zero human intervention. The system monitors consistently without the cognitive fatigue that impacts live supervision.

Multi-Source Input Support: The platform efficiently accepts local videos, RTSP cameras, and YouTube live streaming. Traffic authorities can directly plug in their current IP camera feeds without upgrading infrastructure.

Temporal Confirmation via SORT: The tracker algorithm verifies the continuity of the collision before acting. This effectively prevents the random system false alarms that degrade trust in automated solutions.

Scalable Architecture: Since the Flask backend and React.js frontend operate independently, future updates are straightforward. We can upgrade the YOLOv8 model directly without disrupting the dashboard.

## IX. LIMITATIONS

The current system operates effectively, but there are certain environmental constraints.

Environmental Sensitivity: Extreme weather systems like heavy fog and thick rain can impact visual clarity. Upgrading the physical cameras to include modern infrared technology would easily mitigate this issue.

Rare False Positives in Dense Traffic: Extreme bumper-to-bumper car traffic can occasionally confuse the bounding boxes. Although these quickly resolve, an occasional temporary overlap might trigger a minimal tracking error.

Dependency on Camera Quality: Poor video streams with heavy digital compression drop the available frame quality. The system realistically requires at least a clear 15 frames per second visual feed.

## X. FUTURE SCOPE

The current software serves as a stable foundation for further development.

1. Integration with Civic CAD Systems: Incorporating the notification system into local emergency dispatch software would eliminate the manual email reading requirement.
2. React Native Mobile App: Emergency responders could directly access live dashboard data, maps, and severity ratings from their mobile smartphones while en route.
3. Accident Severity Classification: A feature distinguishing between minor crashes and significant multi-car collisions would help operators dispatch the correct medical resources immediately.
4. Smart City API Ecosystem: Connecting the project to dynamic highway variable message signs (VMS) could visually alert incoming drivers of accidents immediately.

## XI. CONCLUSION

Road accidents will unfortunately remain a significant reality for the foreseeable future. However, this project clearly demonstrates that technology can heavily compress the initial required response time. By combining the YOLOv8 object detection model with the SORT multi-object tracking algorithm wrapped in a modern web architecture, we reduce communication delays from several minutes to just a few seconds.

The system handled different formats properly, seamlessly streaming local files, RTSP feeds, and YouTube live broadcasts with high precision. The software architecture is cleanly scalable, which ensures newer machine learning models and tracking logic can conveniently be adapted.

Most importantly, this project provides a tangible, functional deployment ready for testing. It actively connects to a MongoDB database, dispatches real email alerts, and maintains a clean React.js interface. The project bridges the complex gap between theoretical computer vision and a concrete practical application.

## XII. REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788.

[2] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," in Proc. IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 2016, pp. 3464-3468.

[3] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," GitHub Repository, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[4] P. Singh, R. Dwivedi, and A. Kant, "Deep Learning Approaches for Real-Time Highway Traffic Surveillance and Incident Detection," Journal of Intelligent Transportation Systems, vol. 24, no. 5, pp. 502-517, 2020.
[5] T. Wang, M. Chen, and L. Zhang, "Edge-Computing Pipeline for Offline Road Accident Analysis Using Convolutional Neural Networks," IEEE Transactions on Intelligent Vehicles, vol. 6, no. 3, pp. 412-421, 2021.
[6] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, vol. 25, pp. 120-123, 2000.
[7] yt-dlp Contributors, "yt-dlp: A youtube-dl fork," GitHub Repository, 2023. [Online]. Available: https://github.com/yt-dlp/yt-dlp