AutoAware.

Automatic Car Safety Alarm System

Problem Statement:

SafeZone: Real-time Video Analytics for Industrial Safety

Problem Description:

Industrial Safety is essential for promoting worker safety, preventing accidents, ensuring compliance, and enhancing overall operational efficiency in industrial environments. By leveraging advanced video analytics technologies, it offers a proactive and data-driven approach to industrial safety management. So here the problem is to develop a real-time video analytics tool, that enhances industrial safety by detecting and preventing potential hazards and unsafe situations in industrial environments.

Chosen Solution:

"AutoAware:Automatic car safety alarm system"

Our solution aims to develop an advanced low light object detection system using YOLOv8, a real-time object detection algorithm. Its core purpose is to enhance driver's safety by identifying potential obstacles and hazards in low light conditions, specifically vehicles and pedestrians. The solution is versatile and applicable across industries such as logistics, delivery, construction, mining, and agriculture. By implementing the system on a driver's perspective video feed, it simulates real-world scenarios, offering practical safety enhancements for various transportation sectors.

Requirement Analysis:

1. *Functional Requirements:*

- **Object Detection:** We have trained a custom YOLOv8 model to achieve real time detection of pedestrians and cars within video streams.Detected objects are displayed with bounding boxes on the video frames.
- **Video Sources:** The web-app supports both live webcam feeds and uploaded video files as input sources.
- **Alarm System:** An integrated alarm system has been developed and integrated, capable of promptly alerting the drivers upon the detection of obstacles within the designated region of interest.
- **User Interface:** The user interface presents the live video feed, complete with bounding boxes around identified obstacles. The interface visually highlights the predefined ROI and provides clear indicators when obstacles enter this area. Auditory alerts are accompanied by corresponding visual cues on the interface.

2. *Environmental Requirements:*

Hardware:

- Minimum processor specs: i3 6th gen.
- Minimum 4 GB RAM.
- Intel integrated graphics.
- Minimum 10 GB disk space.
- Webcam.

Software:

- Python 3.8 or higher is required for proper functionality.
- Other libraries are Flask, OpenCV, playsound, torch, ultralytics etc.
- The web app is compatible with commonly used web browsers such as Chrome, Firefox, Safari, JAVA 8 and Microsoft Edge.
- Operating system : Windows 7 or higher (Recommended)

3. *Constrains and Future Enhancement:*

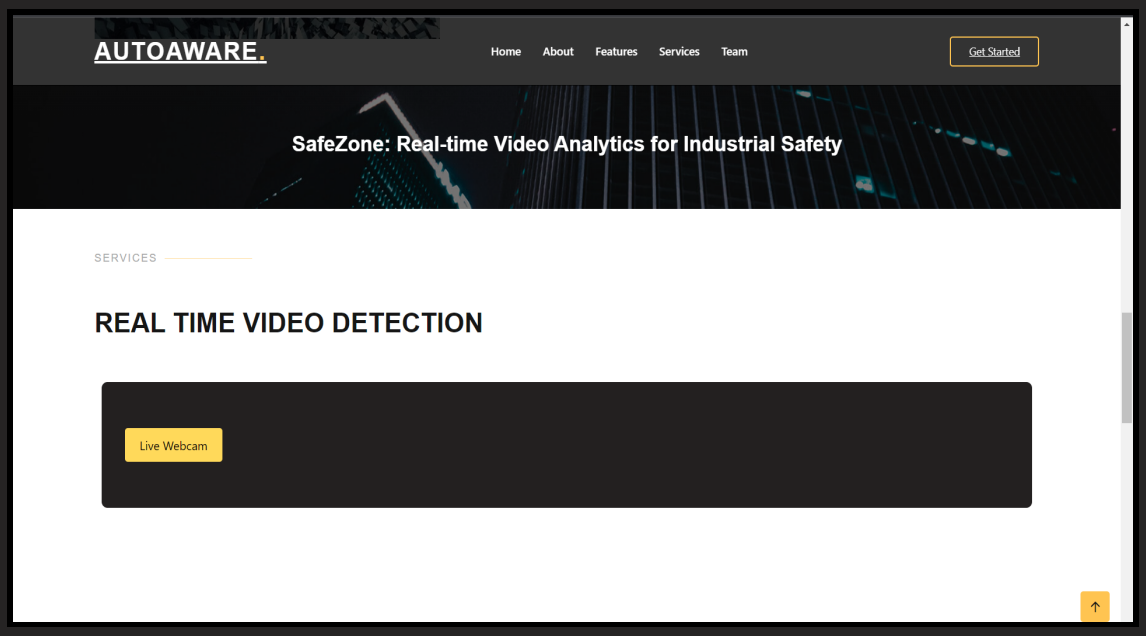
- **Integration with vehicle Systems:** The potential for integrating the system with existing vehicle monitoring and driver assistance technologies has been

considered for future development.

- **Multi-Camera Support:** We're exploring the expansion of the system's capabilities to encompass multiple camera feeds, providing a comprehensive view.
- **Machine Learning Improvements:** We're committed to continuous research and improvements to the YOLOv8 algorithm, aiming to enhance its accuracy and efficiency over time.

System Specification:

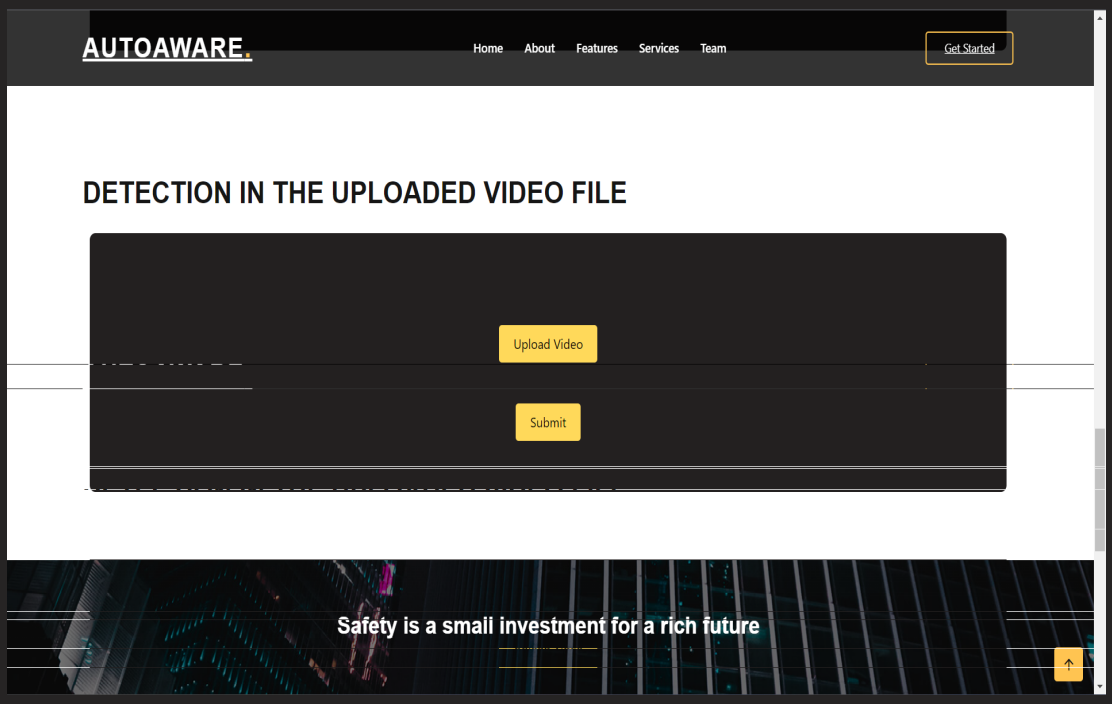
- **Live Webcam based Object Detection:** This feature constitutes the core of the low light object detection web app, enabling real-time identification and tracking of obstacles in the field of view of a webcam. This feature leverages cutting-edge computer vision techniques, including the YOLOv8 algorithm, to instantaneously process video frames from a webcam feed, detect pedestrians and cars, and provide visual cues for enhanced driver awareness. Detected pedestrians and cars are marked with bounding boxes directly overlaid on the live video feed. These bounding boxes outline the positions and dimensions of the identified obstacles. A predefined Region of Interest (ROI) within the video frame is monitored for obstacle entry. When detected objects enter this area, visual cues



are triggered to highlight their presence within the crucial zone. The feature operates in real-time, providing instant feedback to the driver about obstacles

and hazards as they emerge in the webcam's view.

- **Detection based on Uploaded Video File:** This feature is a fundamental aspect of the low light object detection web app, allowing users to analyze pre-recorded videos for object detection and hazard assessment. The uploaded video file is played back within the app's interface, frame by frame, enabling users to visualize the object detection process. This feature extends the application's functionality beyond real-time webcam feeds, enabling users to upload video files and receive valuable insights about potential obstacles, such as pedestrians and cars, in low light conditions.



- **Vehicle Collision Alerting System:** This is a pivotal feature of the low light object detection web app, designed to enhance driver safety by detecting potential collisions and issuing timely alerts. Leveraging real-time object detection capabilities, the system identifies pedestrians and cars in the vicinity of the driver's vehicle, predicts collision risks, and triggers auditory and visual alerts to mitigate the possibility of collisions. The feature acts as an additional layer of safety, preventing potential collisions by alerting the driver before a hazardous situation unfolds. Auditory and visual alerts draw the driver's attention to potential collision risks, helping them remain focused on the road and surroundings.

Technology Stack:

Front-End:

- **HTML:** Used for structuring the user interface and defining the layout of web app components.
- **CSS:** Employed for styling the user interface, applying visual aesthetics, and ensuring a pleasant user experience.
- **JavaScript:** Integrated to create dynamic interactions, handle user inputs, and enhance user engagement within the web app.

Back-End:

- **Flask:** Flask handles routes, manages server-side functions, and connects the front-end with the back-end.
- **Python:** Python plays a pivotal role by serving as the primary programming language that orchestrates various components, libraries, and functionalities to create the low light object detection web app. The use of Python, in conjunction with specific libraries, empowers the project to achieve real-time object detection, user interface management, deep learning model implementation, and audio feedback.
- **PyTorch:** PyTorch is a crucial component in this project, primarily used for the implementation and training of the YOLOv8 object detection algorithm.PyTorch's deep learning capabilities and versatility enable the successful implementation, training, and optimization of the YOLOv8 algorithm, which forms the core of the low light object detection web app's functionality.
- **IBM Watson Machine Learning:** IBM Watson Machine Learning was employed for a critical aspect of data preprocessing, which involved converting object labels into the YOLO format. This format is essential for training the YOLOv8 model effectively, ensuring accurate detection and localization of objects in low light conditions. The platform's tools facilitated the conversion process, optimizing the dataset for subsequent model training and enhancing the overall accuracy of the object detection system.

- **YOLOv8:**
  - The YOLOv8 custom model training process involved several key steps, and it was conducted using the ExDark dataset.The ExDark dataset, known for its focus on low light scenarios, was chosen for training. This dataset contains images representative of real-world low light situations.
  - The training process focused on specific object classes relevant to the project's objectives. These classes included bicycle, bus, car, cat, dog, motorbike, and people.
  - Images and annotations from the ExDark dataset were preprocessed to ensure object annotation format.We had converted the annotation to yolo format.
  - The training process was conducted in a GPU environment, leveraging the computational power of GPUs to significantly expedite the training process.The training of the YOLOv8 custom model was significantly streamlined and enhanced through the utilization of the Ultralytics library. It was done with 50 epochs and the batch size was 8.
  - The model's performance was evaluated using metrics such as Mean Average Precision (mAP). The mAP value reflects the accuracy of object detection and localization.The trained YOLOv8 custom model achieved a mAP value of nearly 70%, indicating a high level of accuracy in detecting and localizing the selected object classes within low light environments.The weight file of this custom training is used for future predictions and real time object detections.
- **Computer Vision:** The OpenCV library is instrumental in the project's computer vision aspect, serving as a key tool for image and video processing.
  - Webcam Feed Capture: OpenCV is used to capture live video feeds from the webcam, allowing real-time access to the environment from the driver's perspective.
  - Frame Processing: The library processes individual video frames in real time. Each frame undergoes analysis for object detection and hazard assessment.
  - Object Detection Visualization: OpenCV facilitates the drawing of bounding boxes around detected objects, providing a visual indication of the presence and location of pedestrians and cars.
  - ROI Monitoring: OpenCV enables the definition and monitoring of a predefined Region of Interest (ROI) within video frames. This facilitates tracking objects within a specific critical area.

- **Alerting System:** The playsound library is utilized to generate an auditory alert, providing an immediate audio cue to the driver.The playsound library plays a designated alert sound file, which could be a distinct sound designed to capture the driver's attention.The auditory alerting system works in tandem with visual alerts, such as visual indicators on the user interface, to provide a comprehensive safety notification.

System Setup:

- **Create Virtual Environment:** Set up a virtual environment using a tool like PyCharm to isolate the project dependencies.
- **Clone Repository:** Clone the project's GitHub repository to a local directory within the virtual environment.
- **Navigate to Project Directory:** Use the terminal/command prompt to navigate to the project directory where the repository was cloned.
- **Install Dependencies:** Run the command *pip install -r requirements.txt* to install the required Python libraries and dependencies listed in the requirements file.
- **Run the Web App:** Execute the app.py file by running the command python app.py in the terminal.
- **Access the Web App:** Once the app is running, you'll see output indicating that the server is active. Open a web browser and enter the provided URL (usually http://localhost:5000) to access the web app.

This setup allows you to interact with the low light object detection system and experience its features firsthand

Acknowledgement:

We extend our sincere gratitude to IBM for their support and resources that have greatly contributed to the success of this project. The opportunity to work with their cutting-edge technologies and platforms has been invaluable.

We would also like to express our heartfelt appreciation to all the guides and mentors that have been instrumental in guiding us throughout the project's development. Their insights, expertise, and unwavering support have been crucial in shaping the project and achieving its objectives.

We acknowledge the collaborative efforts of our fellow team members who have worked diligently to bring this project to fruition. Each contribution, no matter how small, has played a significant role in the project's overall success.

Finally, we would like to thank our families, friends, and colleagues for their understanding, encouragement, and motivation throughout this journey. Your support has been the cornerstone of our progress.