Traffic Sign Detection Using YOLO for Autonomous System : Engineering Track

Aarya Gupta Sargun Singh Khurana Abdullah Shujat
{aarya22006, sargun22450, abdullah22013}@iiitd.ac.in

Abstract

This project implements traffic sign detection using YOLOv8, a Deep Learning model for real-time object detection. The model is trained on a traffic sign dataset with preprocessing and augmentation techniques to enhance accuracy. Results show YOLOv8's effectiveness in detecting signs with high precision, highlighting its potential for autonomous driving and road safety applications. '

1. Problem Statement

The implementation of Traffic Sign Detection using YOLOv8 is crucial for enhancing road safety and autonomous navigation. This project aims to develop a real-time detection system capable of accurately identifying various traffic signs from images and videos, leveraging deep learning for precise recognition.

1.1. Scope of the Problem:

- **Input:** Traffic sign images and video streams captured from dashcams, surveillance cameras, or autonomous vehicle sensors. The dataset includes 4,969 samples, categorized into Train, Validation, and Test sets.
- **Output:** Annotated images and video frames with bounding boxes, labels for detected traffic signs, and confidence scores.

1.2. Target Users:

- **Autonomous Vehicles ADAS Systems:** For real-time traffic sign recognition, enabling compliance with speed limits and stop signals.
- **Traffic Monitoring Authorities:** To monitor traffic rule violations using connected CCTV and IoT-based infrastructure.
- **Urban Planners Smart City Developers:** To analyze traffic compliance patterns and improve urban planning.

1.3. System Interface:

The traffic sign detection system will be accessible via a web application, allowing users to upload images and videos for processing. The application will use a backend model powered by YOLOv8 to analyze the uploaded media, detect traffic signs, and return annotated outputs with labeled bounding boxes and confidence scores. This web-based approach ensures ease of use, scalability, and accessibility across different devices.

By leveraging YOLOv8's real-time processing capabilities, this project aims to provide an efficient and scalable solution for traffic sign detection, contributing to autonomous vehicle safety, smart city infrastructure, and transportation research.

2. Related Work

Traffic sign detection and recognition have been extensively studied, with various methodologies developed to improve accuracy and efficiency. Below is a summary of notable approaches, along with key research papers:

2.1. Support Vector Machine (SVM) with Histogram of Oriented Gradient (HOG)

Approach: This method utilizes the HOG feature descriptor to extract features from traffic sign images, which are then classified using an SVM classifier. It is particularly effective in distinguishing traffic signs from other objects based on shape and edge information.

Reference Paper: "Traffic Sign Detection and Recognition Model Using Support Vector Machine and Histogram of Oriented Gradient" [6] by Nabil Ahmed et al. This study presents a model that detects and recognizes Bangladeshi traffic signs from video frames using SVM and HOG, achieving high precision and accuracy.

2.2. Convolutional Neural Networks (CNNs)

Approach: CNNs automatically learn hierarchical feature representations from raw pixel data, making them highly

effective for image recognition tasks, including traffic sign detection.

Reference Paper: "CNN based Traffic Sign Detection and Recognition on Real Time Video" [7] by Deepali Patil et al. This paper proposes a system that captures signboards using a camera installed in a vehicle, processes the images using a CNN, and notifies the user of detected traffic signs in real-time.

2.3. Hybrid Approaches Combining GIS and Machine Learning

Approach: Integrating Geographic Information Systems (GIS) with machine learning techniques allows for the detection and positioning of traffic signs using geo-tagged images and videos.

Reference Paper: "Computer Vision-Based Traffic Sign Detection and Extraction: A Hybrid Approach Using GIS And Machine Learning" [8] by Zihao Wu. This research proposes methods that utilize geo-tagged Google Street View images and GoPro videos to detect and map traffic signs accurately.

2.4. Baseline Models for Comparison

For our project, we will consider the following models as baselines to evaluate the performance of our YOLOv8-based traffic sign detection system:

- **SVM with HOG:** Provides a traditional machine learning approach for traffic sign detection, offering a benchmark for evaluating the effectiveness of deep learning models.
- **CNN-based Detection Systems:** Serve as a standard for assessing the improvements in accuracy and efficiency achieved by newer architectures like YOLOv8.

By comparing our system's performance against these established models, we aim to demonstrate the advancements and effectiveness of our approach in traffic sign detection.

3. Datasets

The system is currently trained on the **Traffic Sign Detection Dataset** [5], which contains images of vehicles and their surroundings. This dataset enables the initial development and validation of the YOLOv8 model, allowing for preliminary testing of sign detection capabilities before integrating more specialized datasets to improve accuracy.

To enhance model performance and better generalization across various environments, additional datasets will be integrated in future which include:

- **GTSRB - German Traffic Sign Recognition Benchmark** [1]
- **Road Sign Detection** [2]
- **Indian Traffic Sign Dataset** [3]
- **Traffic Signs Dataset - Indian Roads** [4]

3.1. Exploratory Data Analysis (EDA)

Key Statistics:

- **Class Distribution:** The datasets exhibit class imbalance, where common signs such as speed limits appear more frequently than rare signs.
- **Image Quality:** Some datasets contain low-resolution images or images with motion blur, requiring preprocessing techniques such as noise reduction and image enhancement.
- **Lighting Variations:** The datasets include images taken under different lighting conditions, necessitating data augmentation techniques.
- **Domain Shift:** The datasets cover different countries, meaning models trained on one dataset may need fine-tuning for another region.

Some peculiarities in the dataset include:

- **Class Imbalance:** Some classes, such as stop signs and speed limits, are well-represented, whereas traffic light signs (Red Light, Green Light) have relatively fewer instances.
- **Domain Shifts:** The dataset contains images captured in different lighting conditions (day, night, fog), which may affect detection performance.
- **Varied Image Quality:** Some images have low resolution or are partially occluded, making recognition more challenging.

3.2. Evaluation Metrics

- **Precision:** 94.2%, with Stop Signs (97.3%) and Speed Limit Signs (92-100%) showing strong results. Red (88.1%) and Green Light (82.6%) had slightly lower precision.
- **Recall:** 90.6%, with Stop Signs (98.8%) and Speed Limits (94.1-99.1%) performing well. Lower recall for Red (69.4%) and Green Light (74.1%) suggests occasional missed detections.
- **mAP@0.5:** 95.7%, indicating strong detection capabilities.
- **mAP@0.5-0.95:** 82.97%, reflecting challenges in precise localization.
- **Inference Speed:** 2.4ms per image, supporting real-time applications.
- **Loss Metrics:**
  - Train Box Loss: 0.49017, Validation Box Loss: 0.53427
  - Train Class Loss: 0.38064, Validation Class Loss: 0.35107
  - Train DFL Loss: 0.90189, Validation DFL Loss: 0.93138
- **Speed Metrics:**
  - Inference Speed: 2.4ms per image
  - Preprocessing Speed: 1.5ms per image
  - Postprocessing Speed: 0.9ms per image

4. Compute Resources

- **GPU**: Kaggle T4x2 (2 Tesla T4 GPUs with 16 GB VRAM each, 32 GB total)
- **CPU**: Standard multi-core CPU (exact model unspecified)
- **RAM**: 16 GB or more
- **Operating System**: Linux (Ubuntu-based)
- **CUDA Version**: CUDA 11.x
- **Python Version**: Python 3.10.x
- **Framework**: PyTorch 2.0 (with GPU support for training YOLOv8 model)
- **Training Configuration**: Batch size- 8, 16, 32, 64 , epochs- 10, 50, 100

5. Analysis of Results

The model demonstrated strong performance in most categories, particularly for Stop Signs and Speed Limit Signs. However, performance for Red Light and Green Light traffic signals was comparatively lower.

5.1. Class-wise Performance

- **Stop Sign:** High precision (97.3%) and recall (98.8%) indicate that the model is excellent at detecting stop signs. The mAP@0.5 (99.4%) and mAP@0.5-0.95 (93.3%) further confirm its accuracy.
- **Speed Limit Signs:** Performance varies based on speed limit categories, but mAP@0.5 scores range from 92% to 100%, making them highly reliable.
- **Red and Green Light:** The recall for Red Light (69.4%) and Green Light (74.1%) is lower compared to other categories. This suggests the model occasionally misses these signs, likely due to variations in lighting and occlusions.
- **Speed Limits (20-120):** Most speed limit signs performed well, but the highest category (Speed Limit 120) had slightly lower precision and recall.

6. System Integration and Deployment

Following the successful implementation and evaluation of the core YOLOv8 traffic sign detection model (detailed in previous sections), the subsequent phase focused on integrating this model into a user-friendly web application and establishing a robust MLOps pipeline for automation and deployment. This involved developing distinct frontend and backend components and leveraging modern tools for containerization and continuous integration/deployment. The complete integrated system is available in the project repository[1].

[1] https://github.com/Apocalypse3007/SignTrack

6.1. Web Application Architecture

A client-server architecture was adopted for the web application:

- **Backend:** A RESTful API was developed using the `Flask` framework in Python. Its primary responsibilities are to handle incoming prediction requests, process images using the YOLOv8 model, and serve the frontend application.
- **Frontend:** A dynamic and interactive user interface was built using the `React` JavaScript library. It allows users to interact with the system by uploading images and viewing the detection results.
- **Communication:** The frontend communicates with the backend via HTTP requests. Image data is typically sent to the backend API (e.g., as a base64 encoded string within a JSON payload), and the backend responds with the processed image and detection details, also usually in JSON format.

6.2. Backend API Implementation (Flask)

The Flask backend (`app.py`) serves as the core logic hub:

- **Model Loading:** The fine-tuned YOLOv8 model (`weights/best.pt`) is loaded into memory upon application startup for efficient inference.
- **Prediction Endpoint:** A primary API endpoint (e.g., `/predict`) was implemented to:
  1. Receive image data uploaded from the frontend.
  2. Decode the image data (e.g., from base64) into a format compatible with OpenCV and the model.
  3. Perform traffic sign detection inference using the loaded `ultralytics` YOLOv8 model.
  4. Process the model's output (bounding boxes, class labels, confidence scores).
  5. Utilize `OpenCV` (`cv2`) to draw the bounding boxes and labels onto the input image.
  6. Encode the annotated image (e.g., back to base64) and structure the results (annotated image, detected classes) into a JSON response.
  7. Send the JSON response back to the frontend.
- **Frontend Serving:** The backend is also configured to serve the static built files (HTML, CSS, JavaScript) of the React frontend application, typically from a designated folder (e.g., a `static` or `build` folder).
- **Production Server:** `Gunicorn` is included as a dependency, indicating its intended use as a production-ready WSGI server for running the Flask application, offering better performance and stability than Flask's built-in development server.

6.3. Frontend User Interface (React)

The frontend, located in the `frontend/` directory, provides the user interaction layer:

- **User Interaction:** It presents an interface (developed using `React` components) allowing users to easily select or drag-and-drop an image file for analysis.
- **API Communication:** Upon user action (e.g., clicking a "Detect" button), the frontend uses JavaScript's `fetch` API or libraries like `axios` to send the selected image data to the backend's `/predict` endpoint.
- **Result Display:** It receives the JSON response from the backend and dynamically updates the UI to display the returned annotated image, clearly showing the detected traffic signs and their bounding boxes.
- **Build Process:** The React code is built into optimized static assets (HTML, CSS, JS) using `npm run build`. These static files are then served by the Flask backend.

6.4. User Interface Testing

The web interface underwent functional testing, verifying key interactions like image selection and annotated result display. The React frontend and Flask backend communication flow worked as expected, providing a responsive user experience for single-image analysis.

6.5. MLOps and Deployment Pipeline

To ensure reproducibility, automation, and ease of deployment, an MLOps pipeline was established:

- **Containerization (Docker):** A `Dockerfile` was created to encapsulate the entire application. This includes:
  - The Python environment with all backend dependencies (`Flask`, `ultralytics`, `opencv-python-headless`, `gunicorn`, etc.).
  - The backend application code (`app.py`, etc.).
  - The pre-trained YOLOv8 model weights (`weights/best.pt`).
  - The *built* static files from the React frontend (`frontend/build`).
  - Configuration to run the application using `Gunicorn` upon container start.

  This creates a self-contained, portable image of the application.
- **Continuous Integration/Continuous Deployment (CI/CD):** GitHub Actions was utilized (`.github/workflows/ci.yml`) to automate the build and packaging process:
  - The workflow triggers automatically on code pushes or pull requests to the main branch.
  - It sets up the necessary Node.js and Python environments.
  - It installs dependencies for both the frontend (`npm install`) and backend (`pip install`).
  - It builds the static React frontend (`npm run build`).
  - It builds the Docker image using the `Dockerfile`, bundling the backend code, frontend build, and model.
  - It pushes the built Docker image to a container registry (e.g., Docker Hub or GitHub Container Registry), making it readily available for deployment.
- **Deployment Strategy:** The output of the CI/CD pipeline is a versioned Docker image. This image can be deployed to various cloud platforms (AWS ECS, Google Cloud Run, Azure App Service, etc.), Kubernetes clusters, or any server environment supporting Docker containers.

This integrated system provides a complete solution, from user interaction via a web interface to automated building and packaging for deployment, significantly enhancing the usability and maintainability of the traffic sign detection model.

7. Next Steps and Future Work

7.1. Current System Assessment

- **Strengths:** The integrated system successfully provides traffic sign detection via a web interface, leveraging a YOLOv8 model with high precision (94.2
- **Areas for Improvement:** Key areas requiring attention remain the model's performance on Red/Green traffic lights (lower recall) and overall localization precision (mAP@0.5-0.95: 82.97

7.2. Planned Enhancements

With the core application structure in place, future work will focus on:

- **Model Refinement:** Continue optimizing YOLOv8 hyperparameters and potentially fine-tuning the architecture to improve accuracy, particularly for traffic lights and precise bounding box localization.
- **Data Strategy:** Actively expand the training dataset with more diverse scenarios (various lighting, weather, partial occlusions) focusing on under-represented classes. Implement advanced data augmentation techniques.
- **Real-time Video Processing:** Enhance the web application to support real-time detection from video streams (e.g., webcam or uploaded video files), improving UI responsiveness for continuous feedback.
- **Deployment and Monitoring:** Deploy the containerized application to a target environment (e.g., cloud platform). Implement monitoring for performance and resource usage.
- **Robustness Testing:** Conduct thorough testing using diverse real-world image and video sources to evaluate the model's generalization capabilities and robustness in different conditions.

Conclusion: The integrated system provides a solid foundation. Future efforts will prioritize enhancing the core model's accuracy for difficult cases, expanding data diversity, enabling real-time video analysis, and validating performance in realistic scenarios.

8. Member Contribution

Contributions were distributed across the project lifecycle, ensuring involvement in both machine learning and system integration aspects:

- **Aarya Gupta**:
  - ML: Model Training, Performance Evaluation & Analysis, Exploratory Data Analysis (EDA).
  - Full Stack/MLOps: Backend API Development (Flask), Containerization (Docker).
- **Abdullah Shujat**:
  - ML: Dataset Collection & Sourcing, Data Preprocessing & Augmentation, Literature Review.
  - Full Stack/MLOps: Frontend UI Development (React).
- **Sargun Singh Khurana**:
  - ML: Research (Baselines, Techniques), Initial Model Testing & Debugging.
  - Full Stack/MLOps: System Integration (FE/BE), CI/CD Pipeline Setup (GitHub Actions), UI Testing & Full System Debugging.

References

[1] GTSRB - German Traffic Sign Dataset, https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign 2

[2] Road Sign Detection Dataset, https://www.kaggle.com/datasets/andrewmvd/road-sign-detection 2

[3] Indian Traffic Sign Dataset, https://www.kaggle.com/datasets/neelpratiksha/indian-traffic-sign-dataset 2

[4] Traffic Signs Dataset - Indian Roads, https://www.kaggle.com/datasets/kaustubhrastogi17/traffic-signs-dataset-indian-roads 2

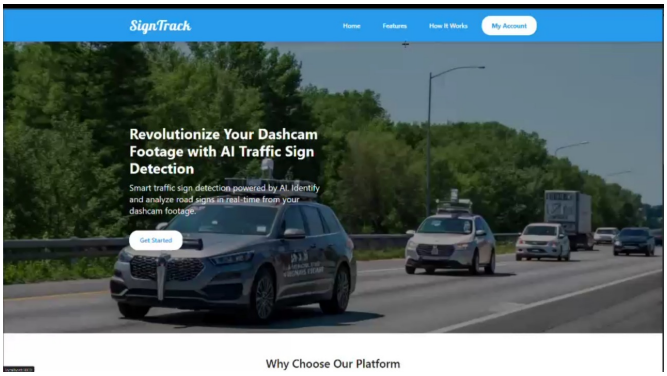
[5] Traffic Sign Detection Dataset, https://universe.roboflow.com/selfdriving-car-qtywx/self-driving-cars-lfjou 2

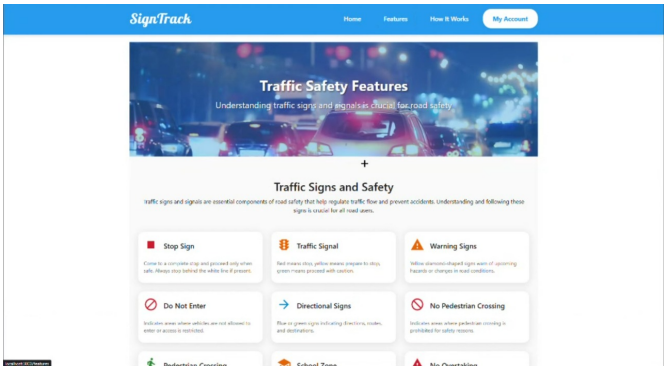
[6] Traffic Sign Detection and Recogni-098 tion Model Using Support Vector Machine and Histogram099 of Oriented Gradient https://www.mecs-press.org/ijitcs/ijitcs-v13-n3/v13n3-5.html 1

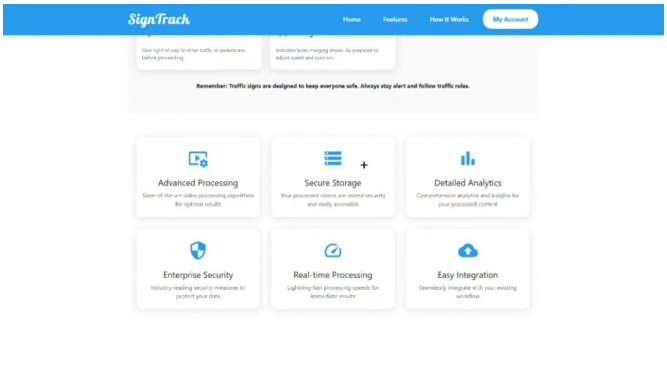
[7] CNN based Traffic Sign Detection https://www.ijert.org/cnn-based-traffic-sign-detection-and-recognition-on-real-time-video 2

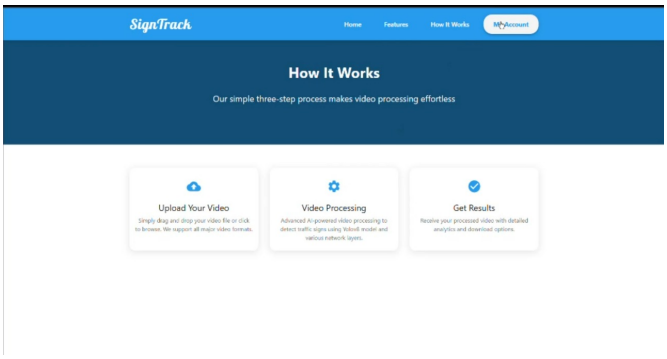
[8] Computer Vision-Based Traffic Sign Detection 2

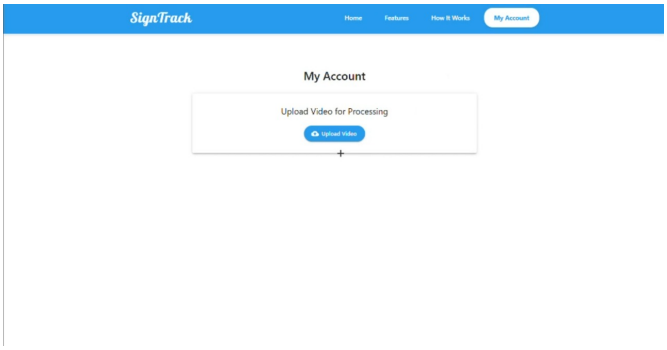
9. Web Application Screenshots (Next Page)

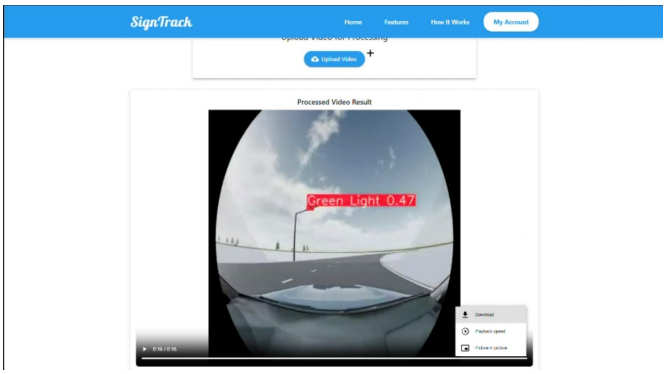

SignTrack
Home
Features
How It Works
My Account
Revolutionize Your Dashcam Footage with AI Traffic Sign Detection
Smart traffic sign detection powered by AI. Identify and analyze road signs in real-time from your dashcam footage.
Get Started
Why Choose Our Platform


SignTrack
Home
Features
How It Works
My Account
Traffic Safety Features
Understanding traffic signs and signals is crucial for road safety
Traffic Signs and Safety
Traffic signs and signals are essential components of road safety that help regulate traffic flow and prevent accidents. Understanding and following these signs is crucial for all road users.
Stop Sign
Traffic Signal
Warning Signs
Do Not Enter
Directional Signs
No Pedestrian Crossing


SignTrack
Home
Features
How It Works
My Account
Remember: Traffic signs are designed to keep everyone safe. Always stay alert and follow traffic rules.
Advanced Processing
Secure Storage
Detailed Analytics
Enterprise Security
Real-time Processing
Easy Integration


SignTrack
Home
Features
How It Works
My Account
How It Works
Our simple three-step process makes video processing effortless
Upload Your Video
Video Processing
Get Results


SignTrack
Home
Features
How It Works
My Account
My Account
Upload Video for Processing
Upload Video


SignTrack
Home
Features
How It Works
My Account
Upload Video
Processed Video Result
Green_Light 0.47