Traffic Sign Detection and Recognition Using YOLO and CNN

Mandala Harsha Yogananda Bharati

July 2025

Abstract

This report summarizes my endterm project on building a complete traffic sign detection and recognition system using deep learning. The project integrates YOLOv8 for detection and a custom CNN for classification, deployed via a Flask web application. The workflow covers dataset preparation, model training, integration, web deployment, and key lessons learned.

1 Introduction

Traffic sign recognition is a fundamental task in intelligent transportation systems and autonomous vehicles. The goal of this project was to detect and classify traffic signs in images using state-of-the-art deep learning models, and to deploy the solution as a user-friendly web application.

2 Dataset Preparation

2.1 Detection Dataset (YOLO)

- Collected images and annotations from the German Traffic Sign Detection Benchmark (GTSDB).
- Converted `.ppm` images to `.jpg` format for YOLO compatibility.
- Used `gt.txt` to generate YOLO-format label files (`class x_center y_center width height`).
- Split data into `train` and `val` sets for model evaluation.

2.2 Recognition Dataset (CNN)

- Used cropped traffic sign images with class annotations.
- Resized all images to 32×32 pixels and normalized pixel values to $[0, 1]$.
- One-hot encoded class labels for multi-class classification.

3 Libraries and Tools Used

- **Python 3.11**: Main programming language.
- **Ultralytics YOLOv8**: For object detection.
- **TensorFlow/Keras**: For CNN-based classification.
- **OpenCV**: For image processing.
- **Flask**: For web application deployment.
- **NumPy, Matplotlib, scikit-learn**: For data handling and visualization.

4 Model Training

4.1 YOLOv8 Detection

- Trained YOLOv8 on the prepared dataset for 32 epochs (due to time constraints).
- Used custom `traffic_signs.yaml` for class names and data paths.
- Achieved reasonable detection accuracy, with bounding boxes drawn around detected signs.

4.2 CNN Classification

- Built a custom CNN with multiple convolutional, pooling, and dense layers.
- Trained on preprocessed traffic sign images for 20 epochs.
- Achieved moderate classification accuracy; further tuning could improve results.

5 Integration and Web Deployment

- Integrated YOLO and CNN: YOLO detects signs, crops are passed to CNN for classification.
- Built a Flask web app with an HTML interface for image upload and display of results.
- The app processes uploaded images, runs detection and classification, and returns annotated images.

6 Results and Evaluation

- **Detection**: YOLOv8 successfully detected most traffic signs in test images.
- **Classification**: CNN classified cropped signs with reasonable accuracy.

- **Web App**: Allowed users to upload images and view detection/classification results interactively.
- Used separate datasets for model evaluation to avoid data leakage[1].

7 Key Learnings

- Importance of correct data preprocessing and annotation format conversion.
- How to train and evaluate deep learning models for object detection and classification.
- Integrating multiple models in a real-world pipeline.
- Deploying a deep learning solution as a web application using Flask.
- Troubleshooting Python environments, library compatibility, and system setup.

8 Conclusion

This project provided hands-on experience in building an end-to-end computer vision system, from data preparation to web deployment. The integration of YOLO and CNN, along with a user-friendly web interface, demonstrates the practical application of deep learning techniques in intelligent transportation.

References

- Ultralytics YOLOv8 Documentation: `https://docs.ultralytics.com/`
- TensorFlow Documentation: `https://www.tensorflow.org/`
- Flask Documentation: `https://flask.palletsprojects.com/`
- German Traffic Sign Detection Benchmark: `http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset`

[1] See memory entry: programming.model_evaluation, 2025-07-01