German Traffic Signs Detection and Classification

Raj Sri Vardhan Ravipati

July 2025

1 Endterm Project: German Traffic Sign Detection and Classification using YOLO and CNN

1.1 Project Overview

The objective of this project was to create a model capable of detecting and classifying German traffic signs from large images. To accomplish this, I implemented a two-stage deep learning pipeline:

1. **YOLOv8n** was used to detect the location of traffic signs within an image.
2. A **custom CNN classifier**, built using `keras.models.Sequential` in TensorFlow, was trained to classify each detected sign into one of 43 classes from the GTSRB dataset.

1.2 Libraries and Tools Used

- **TensorFlow / Keras** – to build and train the CNN classification model.
- **Ultralytics YOLOv8 and PyTorch** – for training and running the detection model using discrete graphics available on the laptop.
- **OpenCV and PIL** for image pre-processing, tiling, and annotation of results.
- **Matplotlib and Seaborn** - to visualize performance metrics and confusion matrix.
- **Flask** – to build a basic web interface to run the entire detection-classification pipeline.
- **Jupyter Notebook & VS Code** – for development, testing, and experimentation.

1.3 Dataset Description

The project used two datasets:

- The **German Traffic Sign Recognition Benchmark (GTSRB)** dataset to train the CNN classifier. It consists of over 50,000 labeled images across 43 traffic sign categories.
- A custom YOLO-formatted dataset containing bounding boxes of traffic signs to train the YOLOv8n detection model.

1.4 Data Pre-processing

For YOLO Detection:

- Images were resized and annotated in YOLO format.
- Large input images were split into 640×640 tiles using a custom tiling script to better detect small traffic signs.

For CNN Classification:

- Images were resized to 128×128 and normalized to the $[0, 1]$ range.
- Data augmentation techniques like rotation, zoom, and brightness changes were applied.
- Labels were mapped using a `label_map.json` file to connect integer class labels to human-readable names.

1.5 YOLOv8 Detection Model

- I used YOLOv8n (nano) from the Ultralytics library for efficient detection.
- The model was trained for 60 epochs on a custom-labeled detection dataset with 43 classes on the gpu-"RTX 4070 Laptop".
- The final model achieved a detection accuracy of **99% mAP50**.

1.6 CNN Classifier (TensorFlow - Keras Sequential)

For classification, I implemented a CNN using `keras.models.Sequential`. The architecture included:

- Three `Conv2D` layers with ReLU activation, each followed by `MaxPooling2D` and `Dropout`.
- A flattening layer followed by fully connected dense layers.
- A final softmax layer for multiclass classification (43 output classes).

- The model was trained using the Adam optimizer and categorical cross-entropy loss.
- Training was performed on CPU for 30 epochs.
- The final validation accuracy reached **98%**, indicating a strong generalization on the clean validation set.

1.7 Integrated Pipeline Architecture

1. The input image is divided into tiles of size 640×640.
2. Each tile is processed by the YOLOv8 model to detect potential traffic signs.
3. Detected bounding boxes are extracted, resized, and passed to the CNN classifier.
4. The CNN returns the predicted traffic sign class, which is then mapped using the label map.
5. The output image is annotated with bounding boxes and class names using OpenCV.

1.8 Deployment using Flask

I developed a Flask-based web interface:

- Users can upload an image through a simple HTML form.
- The backend runs both YOLO detection and CNN classification.
- The resulting image is returned with all signs annotated and labeled.

This experience helped me understand how to manage TensorFlow model loading in production, handle file uploads, and use Flask routing and templating.

1.9 Results and Metrics

- **YOLOv8 mAP50:** 99%
- **CNN Classification Accuracy:** 98% on validation data
- **End-to-End Inference:** Successfully detects and classifies traffic signs even in large images.
- **Visualization:** Confusion matrix, training accuracy/loss plots, Precision, Recall for each class and annotated image outputs.

1.10 Challenges Faced

- Adapting YOLO to handle small objects via tiling.
- Ensuring correct label alignment between YOLO class indices and CNN labels using a `label_map.json`.
- Managing TensorFlow model inference on CPU within Flask without significant delays.
- Avoiding overfitting in CNN through regularization and augmentation.
- Debugging path and file handling issues when integrating Flask, model loading, and image pre-processing.

1.11 What I Learned

This project taught me:

- How to train and fine-tune object detection models such as YOLOv8n on custom datasets.
- How to build and train CNNs from scratch using TensorFlow and Keras Sequential API.
- How to build a real-time inference pipeline by connecting detection and classification stages.
- How to pre-process and tile large images to improve object detection accuracy.
- How to deploy a machine learning model as a web service using Flask and HTML.
- The importance of managing labels, coordinate scaling, and tile offsets correctly.
- Skills in debugging, performance tuning, and writing clean, modular code for large ML projects.

1.12 Conclusion

This end-term project gave me practical, hands-on experience in building a complete deep learning model from scratch. It also helped me clarify the doubt about how AI, other machine learning, and deep-learning models work.