

TDK-dolgozat

Bodai Adrián Tibor

BUDAPEST UNIVERSITY OF TECHNOLOGY
AND ECONOMICS

Faculty of Natural Sciences

**The Development and Evaluation
of Neural Traffic Sign Recognition
Systems**

Author: Adrian Tibor Bodai

BME, TTK, Mathematics Msc

2023/24/1

Supervisor: Dr. Szabolcs Vajna

aiMotive Kft.

Supervisor at BME: Dr. Roland Molontay

Department of Stochastics

Budapest, 2024

Contents

Contents	1
Acknowledgements	3
1 Introduction	4
2 The Data	5
2.1 First stage (YOLO) data sources	5
2.2 aiMotive 3D Traffic Light and Traffic Sign Dataset	5
2.2.1 General description	5
2.2.2 Data transformation	8
2.2.3 Data cleaning and filtering	10
2.3 Roboflow data	15
2.4 GTSRB dataset	16
2.5 Conclusion	17
3 Machine learning algorithms	18
3.1 YOLO Neural Network	18
3.1.1 YOLOv8 Training and Augmentations	18
3.2 Simple Convolutional model	20
4 Task specific benchmarking	22
4.1 Association step	22
4.2 Metric calculation	23
5 Results	24
5.1 Training of the models	24
5.1.1 YOLO trainig	24
5.1.2 Training of the simple convolutional model	27
5.2 Detection Tracking	29
5.3 Evaluation and benchmarking	35
5.3.1 Evaluate frame by frame model outputs	35

5.3.2	Evaluation of Tracked Model Outputs	37
6	Summary	39

Acknowledgements

I would like to express my gratitude to the aiMotive team for providing the essential open source data and for allowing me to use their server capacity to train neural networks. I am especially thankful to my supervisor, Dr. Szabolcs Vajna, who continuously supported my research and provided valuable professional guidance throughout the project.

1 Introduction

In recent years, neural networks have demonstrated significant potential in computer vision tasks, including object detection, text recognition, and data extraction from complex layouts. One of the challenging tasks in this domain is the automated recognition of traffic signs, which proved to be essential on many fields. One such field is the development of self driving cars which could be essential in saving lives and providing a safer method of transfer in the long run. Of course to achieve this we have to develop stable, reliable vision systems which are capable of recognizing the traffic scenario.

A key component in intelligent speed assistance systems is the development of reliable speed sign recognition, which is already an essential feature in modern vehicles. By notifying drivers when speed limits are exceeded, such systems enhance road safety. The development of a robust sign recognition system involves tackling issues such as diverse sign structures, variable font sizes, and non-standard layouts. Traditional methods rely heavily on rule-based approaches, which often struggle with flexibility and scalability. Neural networks, particularly deep learning models, have shown promise due to their adaptability and capability to learn directly from data. However, achieving high accuracy in sign recognition also requires addressing significant data handling, computational and training challenges.

This paper explores the development and evaluation of a neural speed sign recognition system aimed at optimizing recognition accuracy and robustness. The core objective is to design an architecture that has decent recognition accuracy, and to perform a comprehensive evaluation to assess the system's performance on a real world dataset. The paper integrates pre-processing techniques for specific types of data, details of neural network training, and a post-processing tracking method that can improve detection performance. This tracking approach allows the model to achieve relatively strong results even without requiring perfect training accuracy.

2 The Data

In this section, I will describe the datasets used in this paper.

One of the primary challenges in any data science project is acquiring sufficient quality and quantity of data to train a model. For this project, the process is both straightforward and complex, depending on the type of data required.

If we are content with images that are not captured from a vehicle, but rather from low-resolution cameras, a large number of annotated pictures are readily available online. These images can serve as a useful starting point for certain applications.

However, obtaining annotated images captured by high-resolution cameras mounted on vehicles presents a more significant challenge. This kind of data acquisition can be expensive, especially when it involves point clouds gathered by LiDAR (Light Detection and Ranging) technology. If the goal is to train a model in three dimensions, such data is essential and unavoidable [11].

In this paper we investigate how to combine a 2D or 3D detector trained on a limited amount of high-resolution, good quality 3D-annotated data collected from well-calibrated sensors and a diverse set of publicly available 2D traffic sign crop datasets.

2.1 First stage (YOLO) data sources

For this stage, we require high resolution images taken with a large field-of-view sensor, preferably using the same setup as will be used in the deployment of the model to reduce potential domain-gaps.

2.2 aiMotive 3D Traffic Light and Traffic Sign Dataset

In this section I will introduce and analyze the self-driving car dataset created and published by aiMotive Kft.[8][7][5]

2.2.1 General description

The aiMotive 3D Traffic Light and Traffic Sign Dataset provides accurate 3D bounding box annotations for traffic management objects such as traffic lights and road

signs, crucial for autonomous driving systems. These annotations are available for objects up to 200 meters from the ego vehicle, which is essential for vehicles to navigate safely through intersections and urban environments. A key feature of this dataset is that it uses RGB images and GNSS/INS data, without requiring LiDAR point clouds, making the data collection process more accessible while still achieving high accuracy in 3D annotations.

The dataset, approximately 107 GB in size, is organized into various operational design domains (ODDs) like

1. urban - standard, ideal conditions, many objects
2. night - good for lighting robust models
3. rainy - poor visibility
4. highway - completely distinct conditions

It contains sequences of 15-second-long recordings with synchronized sensor data including calibration, camera images, and GNSS/INS data. With this kind of data format, we can also benchmark speed signs instances passed by the ego-vehicle, which is the goal of this paper. Each section has 4 mounted cameras: 2 front cameras (one for closer objects and one with a bigger zoom for detecting distant objects) and one on the left and right side of the car as shown on the figures 1 and 3.

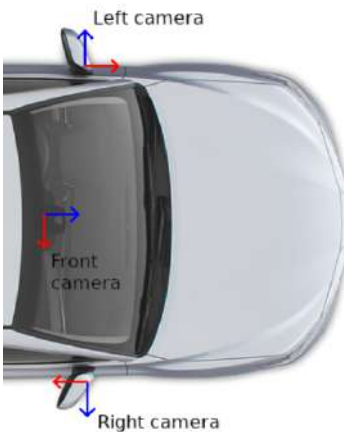


Figure 1: Sensor layout



Figure 2: Body coordinate system



Figure 3: Example images and annotations from an urban section

This dataset is free to use for non-commercial research such as this paper.

2.2.2 Data transformation

For the next step we need to project the 3D object coordinates given in the 3d.body folder to the images themselves, because this dataset only contains the 3D coordinates of speed signs in the ego vehicle's coordinate system.

(This process heavily relies on OpenCV transformations[1].)

To project 3D coordinates onto a 2D image, we utilize the camera's intrinsic and extrinsic matrices which are defined in the calibration folder for each section. The intrinsic matrix K captures the camera's internal parameters, such as focal lengths and principal point coordinates, while the extrinsic matrix, composed of the rotation matrix R and translation vector T , transforms coordinates from the world frame to the camera frame.

The projection can be mathematically expressed as:

$$P_{2D} = K \cdot [R|T] \cdot P_{3D} \quad (1)$$

where P_{3D} is represented in homogeneous coordinates as:

$$P_{3D} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The steps involved in the projection process are as follows:

The intrinsic parameters of the camera are defined as follows:

- f_x : Focal length in the x-direction (in pixels).
- f_y : Focal length in the y-direction (in pixels).
- c_x : x-coordinate of the principal point (usually the center of the image).
- c_y : y-coordinate of the principal point (usually the center of the image).

The mathematical projection can be summarized in the following algorithm:

1. Define the intrinsic matrix K :

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

2. Define the extrinsic matrix $[R|T]$:

$$[R|T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

where $r_{i,j}$ are the elements of the rotation matrix, and t_x, t_y, t_z are the components of the translation vector.

3. Compute the 2D points using the projection formula, resulting in homogeneous coordinates $P_{2D_homogeneous}$.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2)$$

This can be expanded to:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3)$$

where

u: The x-coordinate of the projected 2D point in the image plane.

v: The y-coordinate of the projected 2D point in the image plane.

w: The homogeneous coordinate, which is the scaling factor used in homogeneous coordinate systems. (Depth in this case)

4. Normalize the homogeneous coordinates to obtain 2D image coordinates, so that $w = 1$:

$$P_{2D} = \frac{P_{2D_homogeneous}}{P_{2D_homogeneous}[2]}$$

5. Convert these coordinates to pixels

The projected 2D coordinates (u, v) can then be visualized on the image [Example in Figure 6, 3]. This approach is essential for tasks such as object detection and scene understanding in autonomous driving systems.

2.2.3 Data cleaning and filtering

The accurate recognition and tracking of speed sign objects heavily depend on well-prepared training data. Therefore, our initial task was to clean and preprocess the dataset to ensure it was suitable for training.

The dataset contains all objects within a 200-meter range, including partially obscured ones. To filter out occluded objects, we utilized the "Occluded" attribute in the annotation data, which indicates the visibility of each object in a given frame. We applied a maximum "Occluded" value of 0.05 as the threshold. This value was low enough to exclude nearly all obscured annotations, while still retaining the majority of visible traffic sign objects. Additionally, we discarded annotations that were too small, excessively large, or non-square in shape, ensuring the final dataset remained clean and focused.

Finally, we filtered the data to retain only the relevant classes representing speed signs used in Hungary: (These values can subsequently be mapped to their corresponding classes in the labeled images for visualization purposes.)

- | | | |
|----------------------------|----------------------------|------------------------------|
| 0. <i>eu_speedlimit_5</i> | 5. <i>eu_speedlimit_50</i> | 10. <i>eu_speedlimit_100</i> |
| 1. <i>eu_speedlimit_10</i> | 6. <i>eu_speedlimit_60</i> | 11. <i>eu_speedlimit_110</i> |
| 2. <i>eu_speedlimit_20</i> | 7. <i>eu_speedlimit_70</i> | 12. <i>eu_speedlimit_120</i> |
| 3. <i>eu_speedlimit_30</i> | 8. <i>eu_speedlimit_80</i> | 13. <i>eu_speedlimit_130</i> |
| 4. <i>eu_speedlimit_40</i> | 9. <i>eu_speedlimit_90</i> | |

Note: Keeping the "eu" _speedlimit_x is important to narrow down the classes

to the European standard. The set contains images from the US as well where the design of the signs are completely different, not to mention the units of measure.

We also noticed some wrong annotations, which will likely be corrected in the future, but for now we filtered them manually.

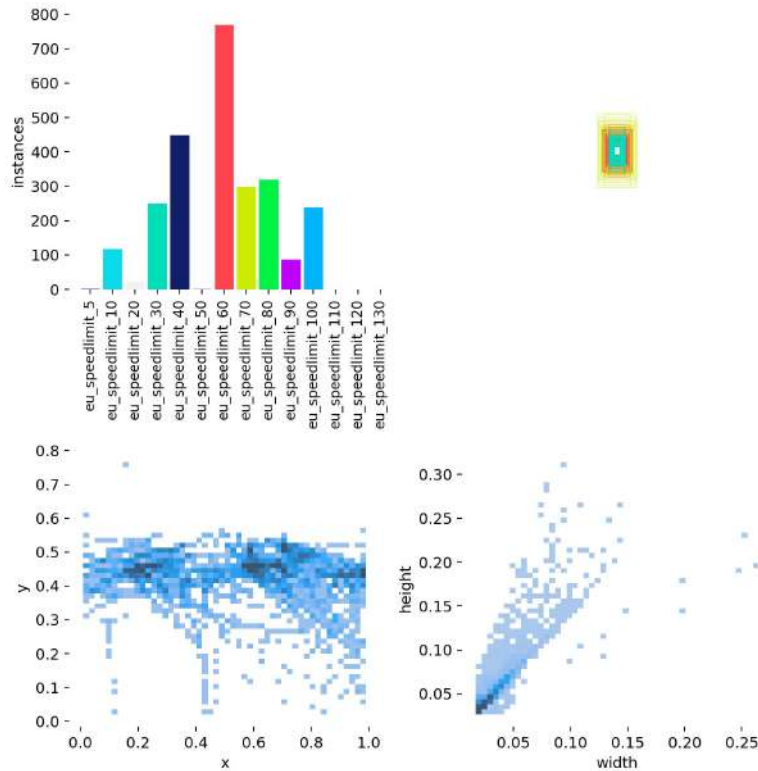


Figure 4: aiMotive Open Dataset class distribution.

Top-left: Object class appearances.

Top-right & Bottom-left: Demonstrate correlations between object dimensions (e.g., width, height) or positions (e.g., center-x, center-y), highlighting common sizes and aspect ratios.

Bottom-right: Distribution of width, height, strong correlation in this case with some anomalies

The final transformed, cleaned dataset had the following distribution over classes (Figure 4) and a training image batch can be seen on Figure 6. We can see, the distribution of speed sign classes highly imbalanced and the amount of instances in the dataset is also not enough to make a really accurate YOLO Deep Learning model. In the end I gathered 3244 images with 2968 speed sign instances (some

images did not contain any instances, so that the model learns not to "guess" every single time).

The validation dataset includes all images from the following sections:

1. 20230713-061248-01.14.00-01.14.15@Rikardo
2. 20231002-123438-00.02.30-00.02.45@Sogun
3. 20231002-123438-01.37.15-01.37.30@Sogun
4. 20231006-114522-00.57.45-00.58.00@Sogun
5. 20231006-114522-00.15.00-00.15.15@Sogun
6. 20230713-061248-01.14.15-01.14.30@Rikardo
7. 20231006-114522-00.21.15-00.21.30@Sogun
8. 20231006-114522-00.22.15-00.22.30@Sogun

Each section consists of 225 images, captured at 15 frames per second over a duration of 15 seconds.

These sections include the following amount of instances as show in Figure 5. The car encounters a speed sign in each section, leading to a total of eight passing instances. This is not a big validation set, but sadly we can not really "waste" more sections on validation, because we would lack the training data.

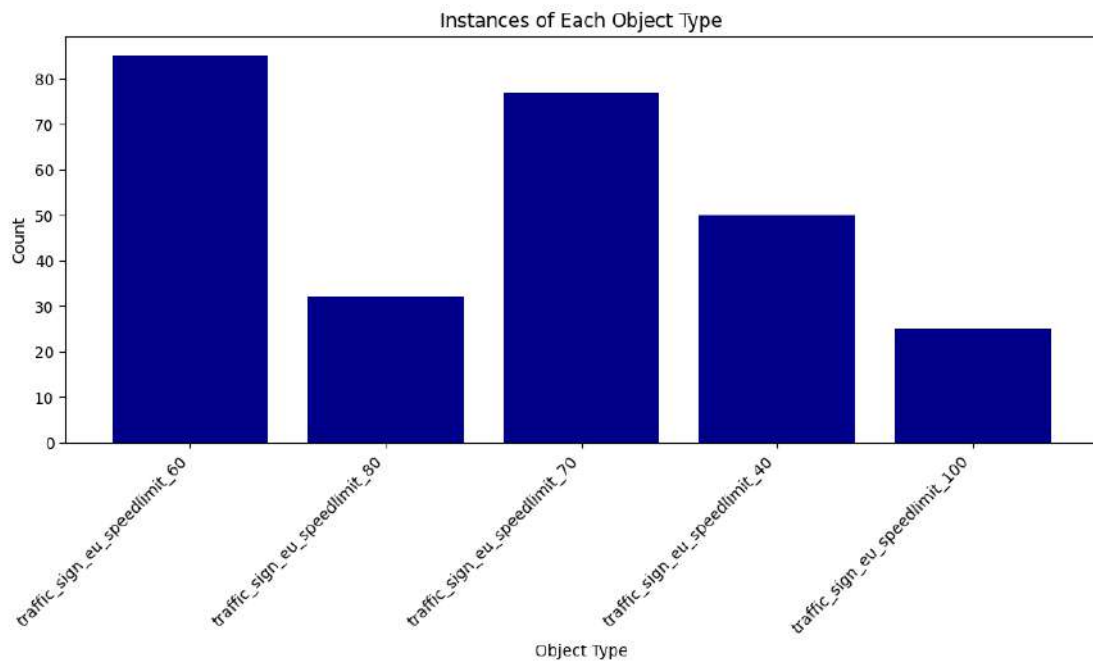


Figure 5: Validation set instance distribution

This alone could produce good models, but we wanted to increase the generalization power of the trained model by adding other datasets to the training data.



Figure 6: Augmented aiMotive open Dataset training batch

2.3 Roboflow data

This dataset [10] is freely available for academic use and currently comprises 4,969 annotated images at the time of writing this paper.



Figure 7: Real world image, not mounted

Figure 8: Image from simulated environment

Figure 9: Not all images are relevant - Stop signs, traffic lights etc.

This dataset was included in the training process to enhance the model's ability to generalize across a wider variety of speed sign objects. The dataset comprises images from multiple countries, including some that are simulated, which helps the model recognize speed signs in less conventional positions. By incorporating diverse data, the model avoids overfitting to Hungarian-specific signs, thereby improving its generalization capabilities.

The distribution of relevant speed sign objects within this dataset is illustrated in Figure 10. While the dataset is relatively balanced and of high quality, it was insufficient on its own as a training set. YOLO models trained exclusively on this dataset performed poorly on the aiMotive Open dataset, which presents a much more challenging validation set.

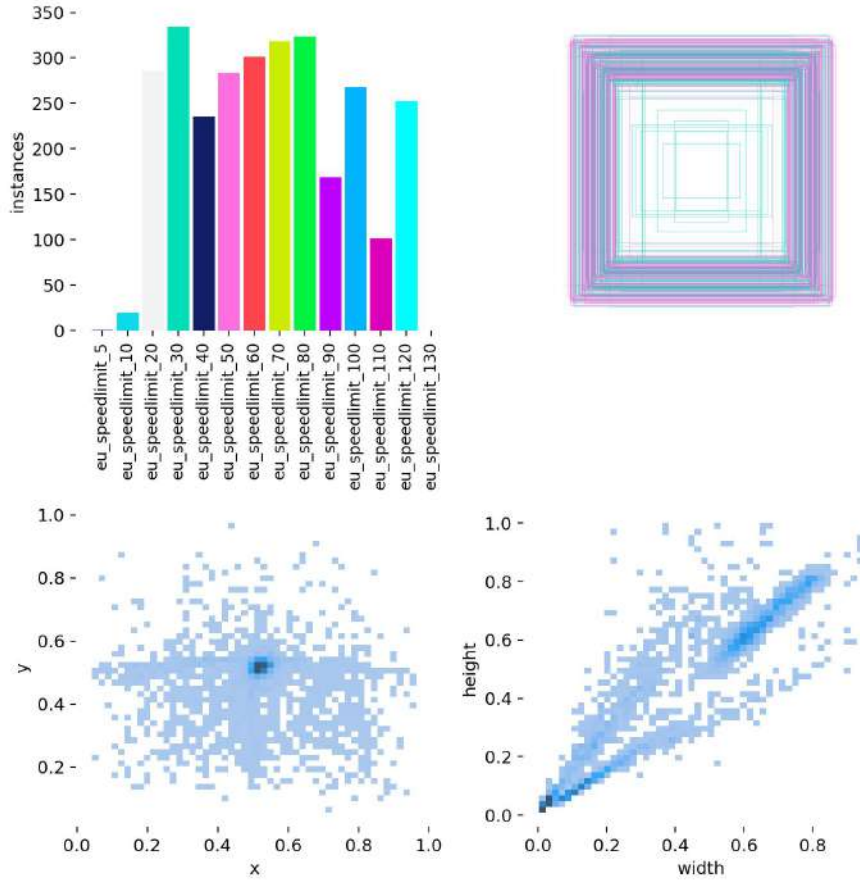


Figure 10: Roboflow Self Driving Cars Dataset relevant classes distribution

2.4 GTSRB dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a widely used resource for training and evaluating algorithms in the domain of traffic sign recognition. This dataset was introduced to facilitate research in the field of computer vision and machine learning, specifically for the recognition of traffic signs in real-world scenarios.

The GTSRB dataset consists of over 50,000 images, which are categorized into 43 different classes of traffic signs. Among these classes, speed signs play a crucial role, as they are essential for maintaining road safety and enforcing speed regulations. The speed signs are specifically categorized into various speed limits. The distribution of speed sign classes within the GTSRB dataset is relatively balanced.

This dataset was only used for the second stage (Simple convolutional model on



Figure 11: GTSRB speed signs

the image crops) model training, to help out the classification head of the YOLO network, because this dataset does not have high resolution pictures, only the image crops are highly available.

2.5 Conclusion

The training dataset was created by combining the three aforementioned datasets. However, the validation set, used to evaluate and select the best model during training, exclusively consisted of images from the aiMotive Open dataset. This decision was driven by the intended application of the model: developing software for self-driving cars that relies only on camera-based predictions. The Roboflow Self-Driving Car dataset, while useful, includes data beyond the scope of mounted camera systems, which are critical for this application.

3 Machine learning algorithms

3.1 YOLO Neural Network

YOLO (You Only Look Once) is a state-of-the-art object detection model that was first introduced by Joseph Redmon et al. in 2016 [9]. It is designed to predict both object classes and bounding boxes in a single forward pass, making it extremely fast and efficient compared to traditional methods that use region proposal-based approaches. YOLO treats object detection as a single regression problem, directly predicting the class probabilities and bounding box coordinates from the input image. This has made YOLO one of the most widely used models in real-time object detection tasks.

The architecture of YOLO is based on a convolutional neural network (CNN), with several versions released over the years, each improving in speed and accuracy. YOLOv8, which I used in this work, offers improved performance by utilizing a more efficient backbone and head, while also supporting a variety of real-time applications. YOLOv8 uses a CSPDarknet backbone and a PANet path-aggregation head, and it follows a single-stage detector design where both classification and localization are done simultaneously.

For an in-depth understanding of YOLO networks, the following references and visual representations provide a good overview: [12] [11] [9]

3.1.1 YOLOv8 Training and Augmentations

During training, I used a YOLOv8 model pre-trained on the COCO dataset [6] to fine-tune the model on the speed sign detection task. The dataset for this task was composed of images containing traffic signs, annotated with bounding boxes and class labels. The following augmentations were applied to the dataset:

- **Fliplr=0.0** and **Flipud=0.0**: These parameters control the probability of flipping the image horizontally or vertically. Since speed signs in real-world images are rarely upside down or mirrored horizontally, I decided not to use these augmentations.

- **Degrees=0.1:** The degree augmentation allows for random rotation of images within the specified range. I used a small rotation value because speed signs are typically seen upright, but allowing a slight rotation helps the model become more robust to tilted camera angles or slight variations in the orientation of the sign.
- **Mixup=0.1:** Mixup is a form of data augmentation where two images are blended together, and their corresponding labels are also mixed. This helps in improving the generalization ability of the model by creating more diverse training samples.
- **Mosaic=0.4:** Mosaic augmentation combines four training images into one during the training process. This allows the network to see objects at different scales and positions, which enhances its ability to generalize across different image compositions and improve detection at various scales.
- **Shear=0.1:** Shear augmentation applies a distortion to the image along the x or y axis. This was applied with a small value to simulate real-world conditions where signs may appear slightly skewed due to camera angles or road curvature.
- **Scale=0.1:** The scale augmentation randomly zooms in or out on the image. This is useful in object detection tasks where the same object can appear at different distances from the camera. A small scaling factor was used to mimic slight changes in the apparent size of the speed signs without dramatically altering the object resolution.

By using these augmentation strategies, I aimed to increase the variability of the training data, enabling the YOLOv8 model to generalize better across different real-world scenarios, such as varying sizes, rotations, and slightly skewed angles of speed signs.

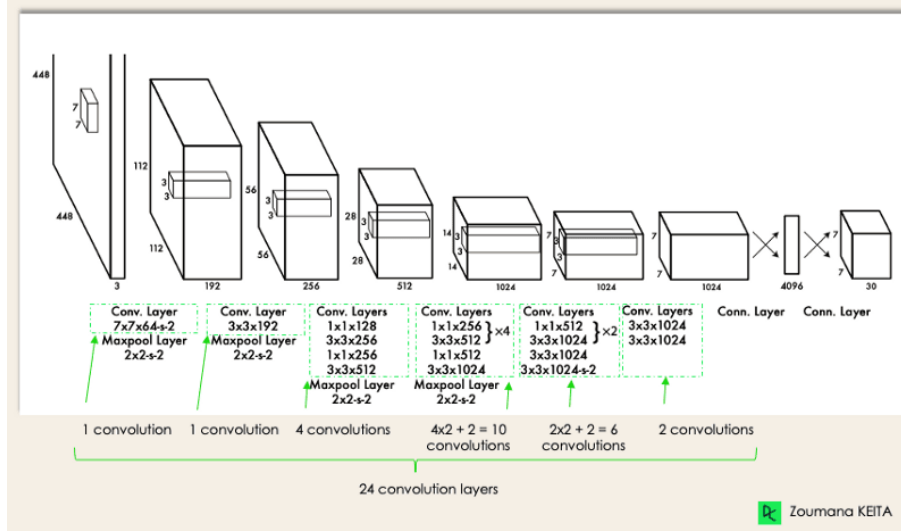


Figure 12: A brief overview of the YOLO Network Architecture Blocks[2]

3.2 Simple Convolutional model

I trained a second model in case the training data was insufficient for the YOLO classification head. This model takes the cropped images from YOLO's suggestions and provides its own classifications. Since this model can be trained on additional data (such as the GTSRB dataset), it may prove useful.

The network architecture consists of a series of convolutional layers, followed by residual blocks using the **BasicBlock** module, and ends with a fully connected classification layer. The overall structure of the network can be summarized as follows:

BasicBlock: A key component of the network is the **BasicBlock**, a residual block designed to facilitate training deeper networks by allowing gradients to flow more easily. Each **BasicBlock** consists of:

- Two 3×3 convolutional layers, each followed by batch normalization and ReLU activation.
- A residual (skip) connection that adds the input directly to the output of the second convolution, helping to preserve information across layers.
- If the input and output dimensions differ, a downsampling layer is applied to adjust the dimensions of the input before adding it to the output.

Table 1: ClassifierNet Architecture Summary

Layer	Output Shape	Description
Conv1	$64 \times \frac{H}{2} \times \frac{W}{2}$	7×7 , Conv2d, Stride 2, Padding 3
BatchNorm1	$64 \times \frac{H}{2} \times \frac{W}{2}$	Batch Normalization
ReLU	$64 \times \frac{H}{2} \times \frac{W}{2}$	ReLU Activation
MaxPool	$64 \times \frac{H}{4} \times \frac{W}{4}$	3×3 Max Pooling, Stride 2, Padding 1
Layer1	$64 \times \frac{H}{4} \times \frac{W}{4}$	2 BasicBlocks, 64 Channels
Layer2	$128 \times \frac{H}{8} \times \frac{W}{8}$	2 BasicBlocks, 128 Channels, Downsampling
Layer3	$256 \times \frac{H}{16} \times \frac{W}{16}$	2 BasicBlocks, 256 Channels, Downsampling
Layer4	$512 \times \frac{H}{32} \times \frac{W}{32}$	2 BasicBlocks, 512 Channels, Downsampling
AvgPool	$512 \times 1 \times 1$	Global Adaptive Average Pooling
FC	10	Fully Connected Layer, Output: 10 Classes

The network follows a ResNet-like structure, using residual blocks (**BasicBlock**) that include two convolutional layers with batch normalization and ReLU activation, combined with a shortcut (identity) connection. Downsampling is applied when transitioning between layers with different numbers of channels.

4 Task specific benchmarking

In this section, I introduce the benchmarking method used to evaluate the models.

I developed a comprehensive package within an existing environment repository named aidrivemetrics, which is owned by aiMotive. As such, I am unable to disclose all details of this benchmark. Notably, the package is compatible with aiMotive’s serialized data and is utilized for commercial applications.

The benchmarking process involves several steps: loading the ground truth data, loading the prediction data, associating the reference and prediction data points, and calculating the metrics specified in the configuration file.

In this paper, the focus is on the association and metric calculation steps, particularly because we aim to evaluate event accuracy—whether the model correctly identifies events (such as recognizing a speed sign as it is passed)—rather than raw frame-by-frame predictions.

4.1 Association step

The association process between reference and predicted data points can be computationally intensive, especially with large numbers of points. Our goal is to pair each reference data point with its nearest predicted data point; however a naive implementation would result in significant time delays over longer sequences.

To address this, we employ the **Hungarian algorithm** in the association step, which efficiently finds optimal matches. This process is outlined as follows:

1. **Data Points and Distance Matrix:** Given n predicted data points and m reference data points in a particular frame, we compute the $n \times m$ distance matrix. Each element d_{ij} in this matrix represents the Euclidean distance between a predicted point p_i and a reference point r_j . In this work, we use Euclidean distances in both 2D and 3D environments, depending on the context.
2. **Squaring Matrix Elements:** To prevent unintentional associations, we square each element of the distance matrix. This squaring ensures good association in cases where prediction biases could otherwise lead to unwanted

pairings. 13

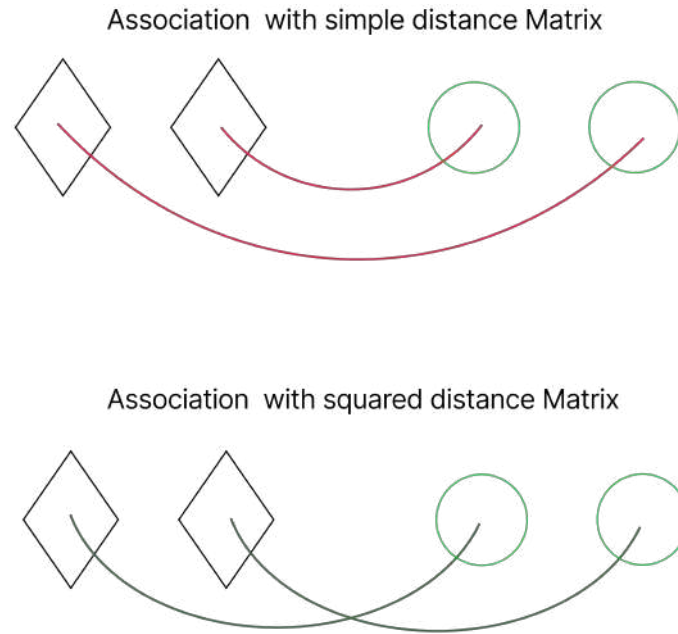


Figure 13: The difference between simple and squared distance when using Hungarian algorithm in case of Bias

3. Optimal Pairing Using the Hungarian Algorithm: Finally, the Hungarian algorithm is applied to the squared distance matrix. This provides the optimal pairing between predicted and reference data points, yielding associated pairs $\{(p_i, r_j)\}$ based on minimal cumulative distances.

The Hungarian algorithm, proposed by Kuhn and Munkres, has a time complexity of $O(n^2m)$, and in cases of square matrices, $O(n^3)$, making it highly suitable for handling larger datasets efficiently [4].

4.2 Metric calculation

There are multiple metrics which will be calculated including **accuracy**, **recall**, **precision** by the default formulas which will not be included in this paper.

5 Results

In this section I will highlight the results achieved in this paper.

5.1 Training of the models

In this section, I will discuss the training dependencies and results.

The networks were trained on a server provided by aiMotive, equipped with a GPU: and CUDA version:

5.1.1 YOLO trainig

The YOLO network was trained using the dataset described in the second chapter. We utilized the combined aiMotive and Roboflow dataset, which proved to be sufficient for our needs. For validation, we selected specific sections from the aiMotive Open dataset.2.2.3

The training metrics are presented in Figure 14.

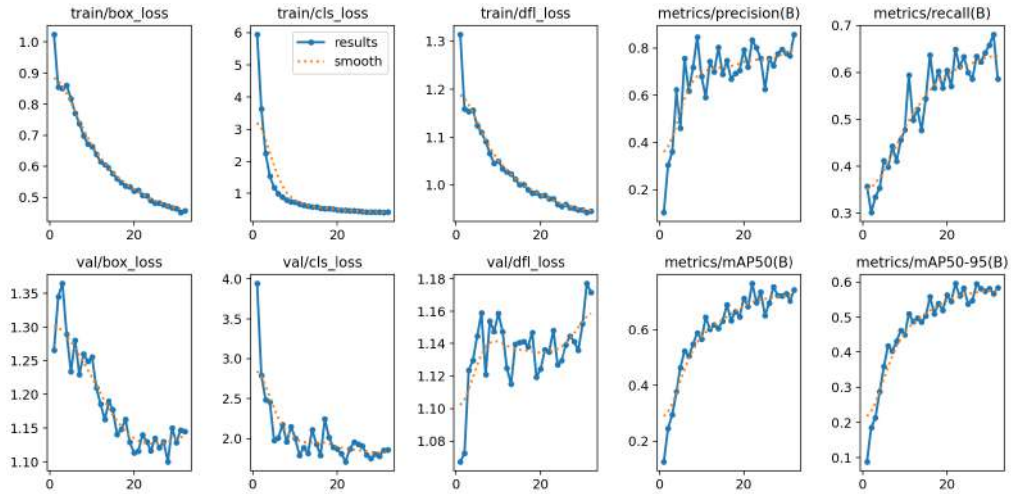


Figure 14: Metrics while training

On Figure 15 we can see some predictions from the final model frame by frame on the validation dataset (which was never seen by the model during training)

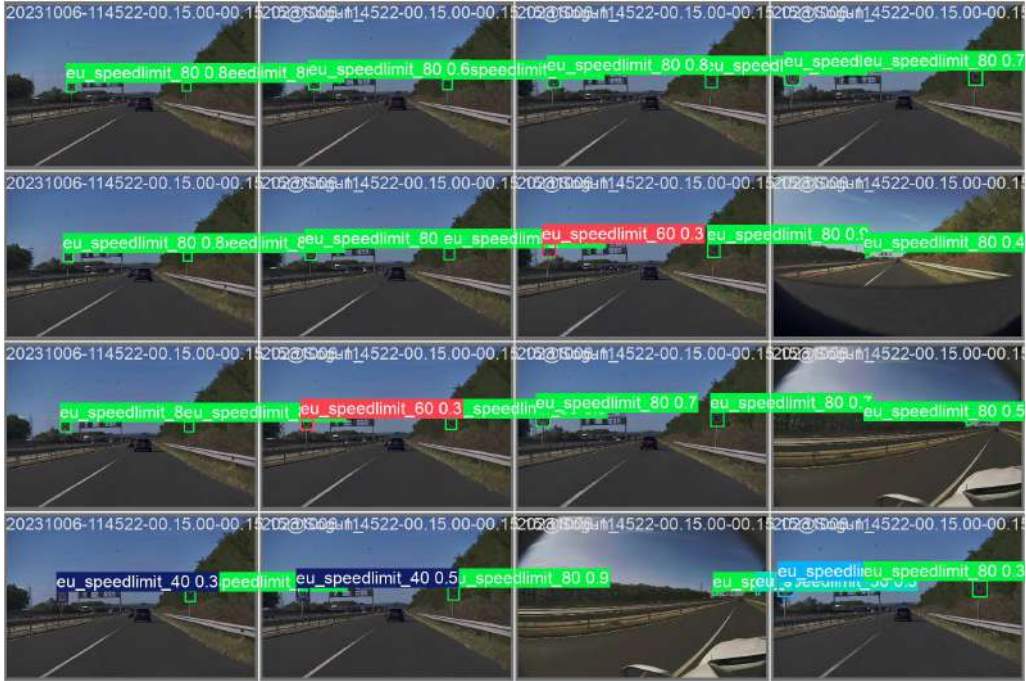


Figure 15: Predictions on the validation set

As observed, the majority of predictions are accurate regarding both the proposed regions and the associated classes. However, there are some misclassifications present in the displayed set. We will try to solve this problem by using the second model on the proposed region, and try to find a threshold cut for both models which satisfy the needs of the task.

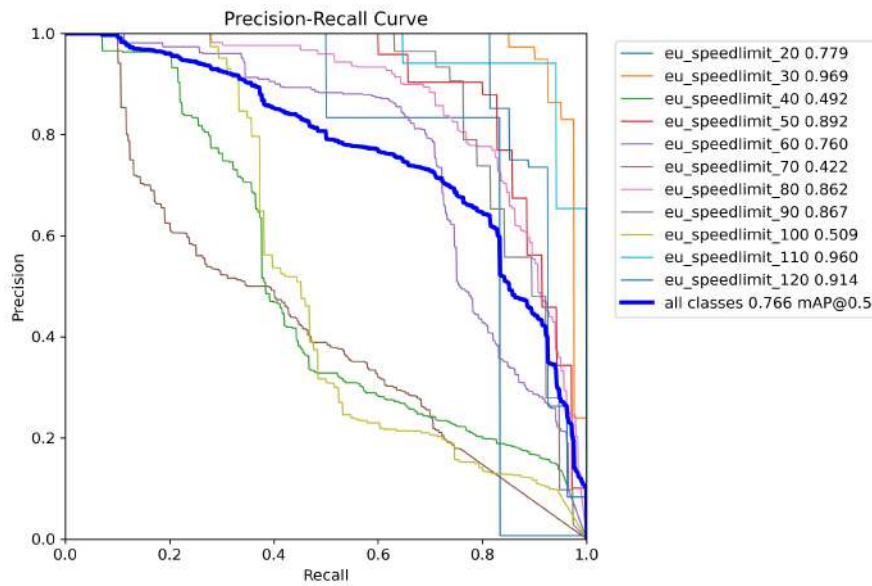


Figure 16: Precision Recall Curve of the YOLO model

In Figure 16, the precision-recall curve is displayed, providing a useful metric for selecting the model's threshold. This plot includes data from Roboflow, not just the original validation data2.2.3 to incorporate a broader range of classes for determining our threshold cut. It's evident that the threshold value can vary significantly across individual classes. This variability can be attributed to the differing number of instances for each class within the dataset.

5.1.2 Training of the simple convolutional model

In this section, we present an overview of the training process for the secondary network, which could serve as a supplementary classifier to the YOLO network’s classification head. This secondary network is employed in cases where the data volume for certain classes is insufficient for effective training within the YOLO framework.

Figure 17 illustrates the instance counts for each class both before and after the augmentation step. Some classes (such as *eu_speedlimit_110*) contain very few images, as they are entirely absent in the GTSRB dataset. Excessive augmentation for these classes would risk overfitting the network, compromising its generalization capability.

The augmentations applied include zooming, rotation, and positional offsets. An example of the augmented images for the *eu_speedlimit_110* class is shown in Figure 17.



Figure 17: Class distribution before and after augmentation (top), and examples of augmented images for the *eu_speedlimit_110* class (bottom).

Figure 18 shows the validation and training accuracy and loss throughout the training process. The final network, which achieved the best performance on the

validation dataset (without augmentation), reached an accuracy of 97.83%. While some of the misclassified validation images could potentially be corrected, most of them are challenging to recognize even for human observers.

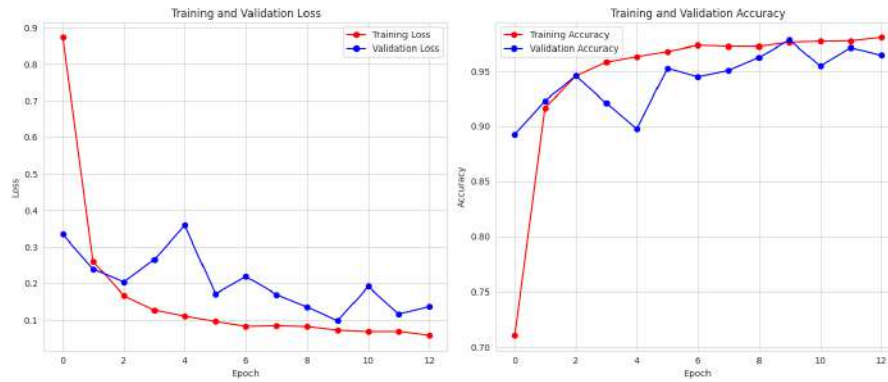


Figure 18: Validation and training loss and accuracy during the training

5.2 Detection Tracking

The algorithm in this section is largely based on the approach in this article [5] about tracking 3D annotations.

Tracking an object over time means that rather than focusing on one frame or timestamp, we observe the surroundings over multiple images and base our assessment on all of the data collected.

In this context, detection tracking implies that we do not rely solely on a single output from the neural network. Instead, we employ a post-processing step that improves the final detections using straightforward algorithms. In this case, we processed data from the last 10 seconds of driving.



Figure 19: Example of the same object from multiple perspectives

To achieve this we transform the detections to a coordinate system which is stationary compared to the objects (speed signs) we would like to detect.

The Earth-Centered, Earth-Fixed (ECEF) coordinate system, as described by Heiskanen and Moritz, is a geocentric system used in satellite and navigation applications. It is particularly valuable in providing a stable, fixed reference point for global positioning and object tracking that accounts for Earth's rotation [3].

Step 1: Project Image Points to Camera Rays The initial detection points, denoted as $\mathbf{p}_{\text{image}}$, are provided in image coordinates (pixel space, where each pixel represents a ray from the camera's perspective, this is a difference from the original algorithm [5], because we do not have the 3 dimensional body coordinates of the detections). These points are transformed into camera rays in the camera's 3 di-

mensional coordinate system, where the camera itself is the zero and the direction which the camera faces is axis X .

This transformation is essential as it allows image-based detections to be projected into 3D space, with each ray originating from the camera center and pointing in the detected object's direction.

Step 2: Transform Camera Rays to ECEF Coordinate System Once the camera rays are defined, they are transformed into the body coordinate system (the coordinate system of the moving car). The transformation from the camera coordinate system to the body coordinate system is represented by the rotation-translation matrix $\mathbf{RT}_{\text{cam-body}}$, which includes a 3D rotation $\mathbf{R}_{\text{cam-body}}$ and a translation vector $\mathbf{t}_{\text{cam-body}}$:

$$\mathbf{r}_{\text{body}} = \mathbf{R}_{\text{cam-body}} \cdot \mathbf{r}_{\text{cam}} + \mathbf{t}_{\text{cam-body}} \quad (4)$$

where $\mathbf{r}_{\text{body}}$ represents the ray in the body coordinate system. This step adjusts for the relative position and orientation of the camera with respect to the vehicle or robot.

The rays are then transformed from the body coordinate system to the ECEF coordinate system, enabling object localization in a stationary coordinate system where all detected speed signs are fixed. Similarly, this transformation uses the rotation-translation matrix $\mathbf{RT}_{\text{ECEF-body}}$, which can also be found in the calibration data of the data source:

$$\mathbf{r}_{\text{ECEF}} = \mathbf{R}_{\text{ECEF-body}} \cdot \mathbf{r}_{\text{body}} + \mathbf{t}_{\text{ECEF-body}} \quad (5)$$

where $\mathbf{r}_{\text{ECEF}}$ is the ray in the ECEF coordinate system.

After this transformation, we obtain rays (lines) in a common stationary coordinate system, as shown in Figure 20.

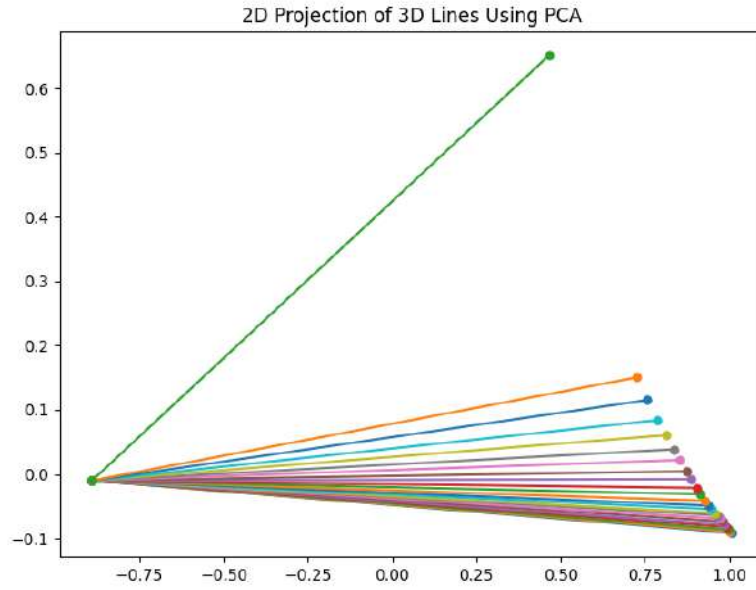


Figure 20: Detection rays in ECEF coordinate system - projection from above using PCA

Each ray corresponds to a detection, so each ray has a class (e.g., *eu_speedlimit_60*). Our trained YOLO model can usually classify detections correctly, but occasionally it may make mistakes. Therefore, we aim to filter out false positives and also localize the 3D position of speed signs in the car's coordinate system so that we know when we pass them.

In Figure 20, the detections converge to a point, representing the target speed sign in 3D space, as all rays are transformed to the same ECEF coordinate system. From this observation, we can derive the following algorithm to estimate the 3D coordinates of detections through triangulation:

Algorithm 1 Triangulate Object Coordinates from 3D Line Intersections

Input: Set of lines $\mathcal{L} = \{\mathbf{l}_1, \mathbf{l}_2, \dots, \mathbf{l}_n\}$ in 3D space, each defined by two points in ECEF coordinates.

Output: Set of estimated object positions $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m\}$ in ECEF coordinates.

Step 1: Project 3D Lines to 2D Space

Apply Principal Component Analysis (PCA) on $\mathcal{L}$ to reduce the dimensionality of the lines from 3D to 2D.

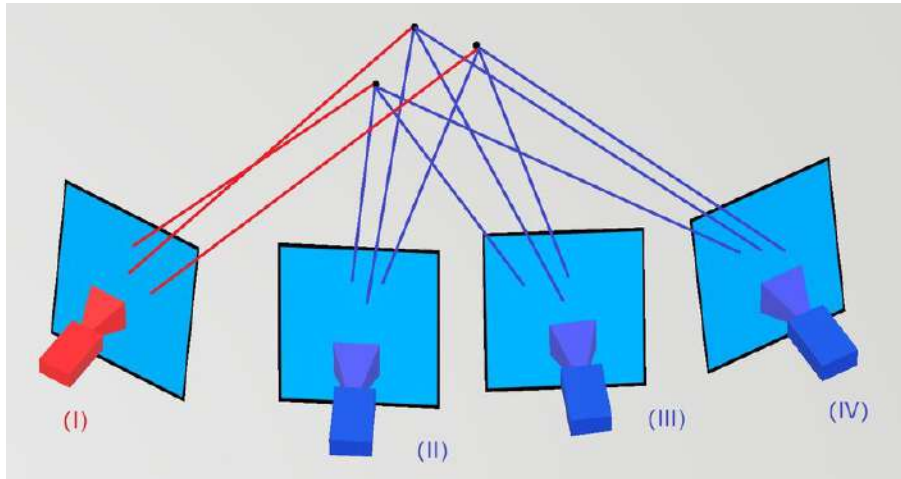
(This also differs from the original algorithm [5])

The purpose of this step is to visualize the detection rays "from above," ensuring intersections. In 3D space, an intersection-based algorithm would not be feasible.

Step 2: Find Intersections of Projected Lines for every object class

We do this step for every object type in the detection space, because we do not want the rays which belong to different type of objects to associate with each other. This is especially good for filtering out lone false positives, which can only be seen for some frames.

These intersections indicate potential speed sign locations. We can accomplish this by leveraging the car's movement, which enables triangulation from multiple perspectives.



Step 3: Cluster Intersection Points in 2D Space

Apply DBSCAN clustering on the intersection points to group them by spatial proximity.

The DBSCAN algorithm effectively filters out false positives and misclassifications, as the likelihood of these events is low given a well-trained model. Outlier points after clustering are likely to represent false detections, while cluster centroids indicate probable detections.

Step 4: Transform Cluster Centers Back to 3D Space

Convert the cluster centers back to 3D space after clustering using the same transformation matrix used during the PCA step.

With this method, we estimate the 3D coordinates of speed signs based on the car's movement and multiple perspectives, while filtering out false detections by class and location. The output of this process is shown in Figure 21.

The DBSCAN parameters were set to `min_samples = 20` and `eps = 0.25`, requiring numerous correct detections from various perspectives within the past 15 seconds to confirm a detection. Given the well-trained model, this approach yields reliable results.

Our final step concludes the detection process: We transform the cluster centers back to the car's coordinate system (The inverse of the same transformation as we used before) and estimate when exactly we are going to pass the speed sign (for example when the coordinate x switches from positive to negative). On the

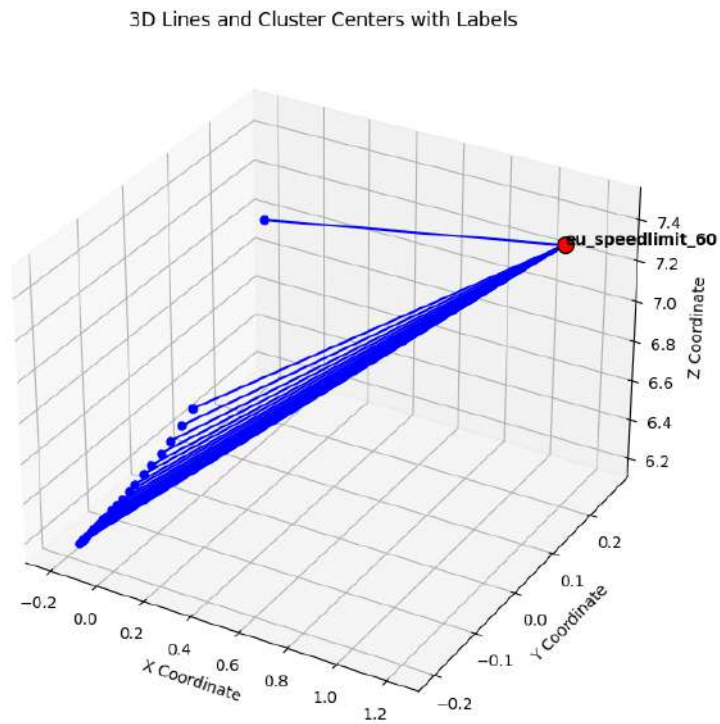


Figure 21: 3D detections based on 2D image space bounding box center rays

5.3 Evaluation and benchmarking

5.3.1 Evaluate frame by frame model outputs

As described in a previous chapter, we implemented a task-specific benchmarking package capable of evaluating both frame-by-frame predictions and tracked detections, in both three-dimensional space and two-dimensional image space. Here, we present the results from this benchmark on the validation set.

First, we present the recall and precision in the case of frame-by-frame predictions, where no detection tracking methods were applied. These benchmark outputs assess our model's ability to draw bounding boxes on the 2 dimensional image and classify detected objects.

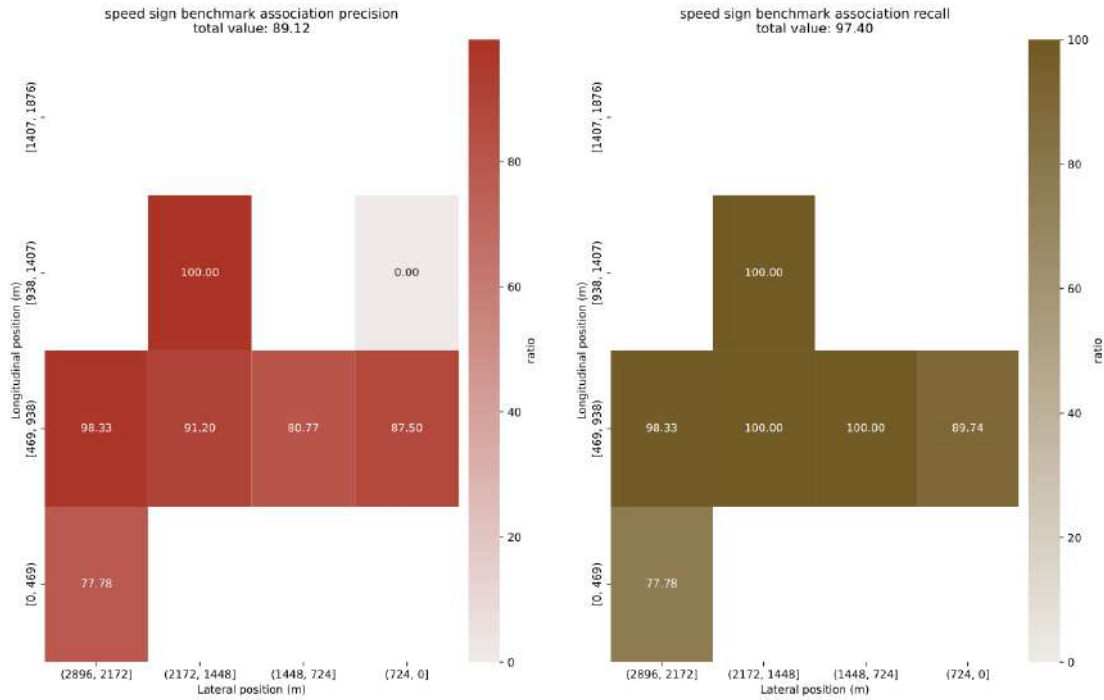


Figure 22: Precision and recall of the ensemble model in image space (evaluated on images with a resolution of 2896×1876 pixels)

These figures22 illustrate the YOLO model's capability to detect speed signs on the image space, achieving a recall of 97.4%, which is commendable given the challenging nature of the validation dataset. The precision of 89.12% indicates that the model occasionally generates bounding boxes where no objects are present or

produces multiple bounding boxes over a single speed sign (which counts as false-positive detections, except for one of these bounding boxes).

The YOLO classification head was also trained effectively; however, we incorporated additionally the simple classifier to create an ensemble model. In cases of disagreement between the YOLO network and the simple classifier, we compare their confidence scores. If the simple classifier’s prediction confidence is at least 1.5 times higher than YOLO’s, we adopt the class prediction of the simple model. With this ensemble approach, we achieved the confusion matrix shown in Figure 23 on the aiMotive validation dataset. (This confusion matrix does not include false positive or false negative cases, it only includes the cases where the model successfully found the bounding box of an object).

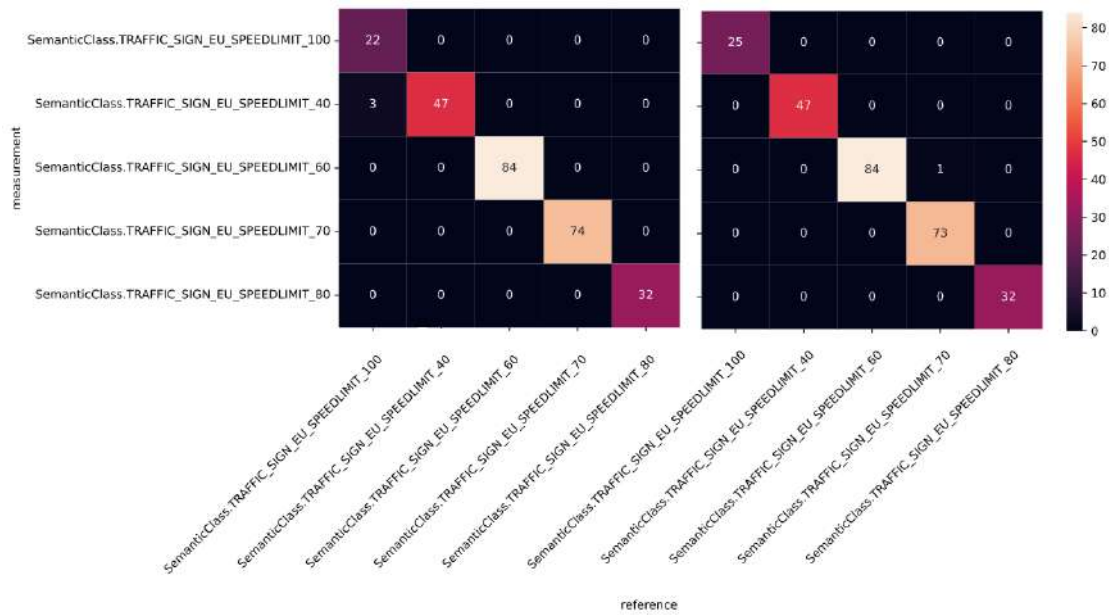


Figure 23: Perception confusion matrix for both the YOLO model (left) and the ensemble model (right)

There were only four instances where the YOLO model’s classification head and the simple classifier disagreed and the confidence criteria was met. In three of these cases, the simple classifier successfully corrected the YOLO model’s prediction; however, in one instance, it resulted in an incorrect prediction. Although this may

appear minor, the ensemble approach reduced classification errors by 66.6%, which could prove highly beneficial in larger datasets.

The following table summarizes the runtime of each network and their combined inference time (tested on a 12th Gen Intel(R) Core(TM) i7-12700H without GPU acceleration to better estimate the runtime on automotive hardware):

Device: 12th Gen Intel(R) Core(TM) i7-12700H (CPU)	
Model	Average Inference Time
YOLO Model	62.3 ms
Simple Classifier (complementary model)	2.9 ms

In summary, to thoroughly evaluate the effectiveness of this ensemble approach, a larger validation and test set would be necessary for a more comprehensive assessment. Additionally, it is essential to consider the perspective of automotive manufacturers, for whom even this level of processing time may be significant in real-world applications.

5.3.2 Evaluation of Tracked Model Outputs

The final evaluation step assesses the model's ability to assist drivers by identifying speed signs as they pass them. For each speed sign encountered, the goal is to generate an event notification. This process is made possible using the aiMotive Open validation dataset, which provides continuous image sequences, enabling the application of tracking methods on YOLO-based detections as previously described.

The results were impressive: across all eight test sections, the model successfully generated an event signal within one second of passing each speed sign. In all cases, the speed limit was accurately identified, as shown in Figure 24.

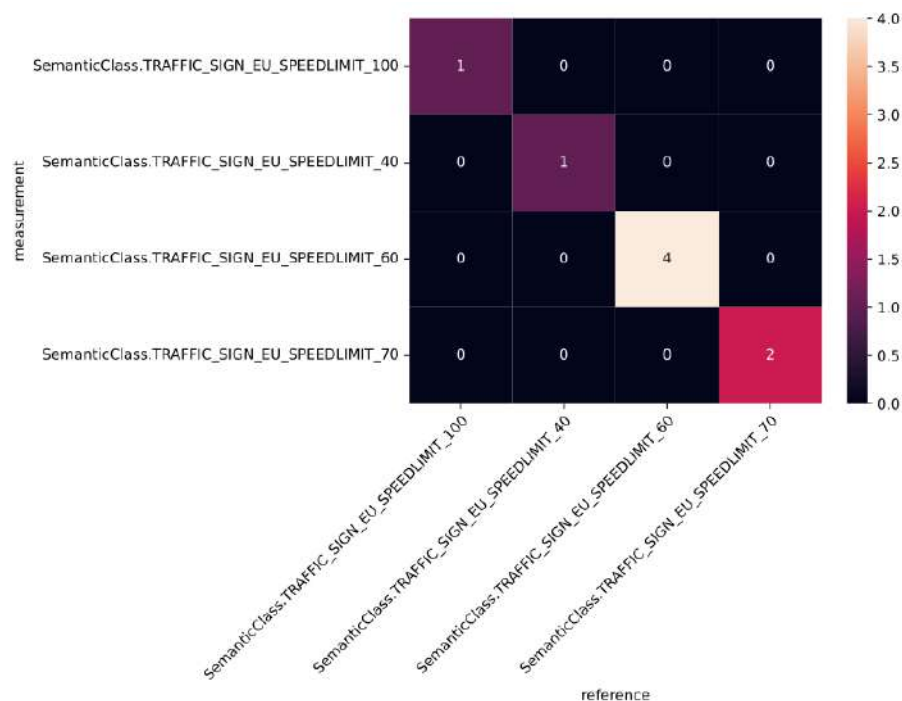


Figure 24: Confusion matrix for tracked event predictions

This result indicates that the system could potentially be deployed in a real vehicle, where the model, along with its tracking system, would reliably identify speed signs as they are passed. However, for real-world application, extensive testing on a larger dataset would be essential to identify weaknesses, and further optimizations would be needed to ensure the system operates efficiently on automotive hardware.

6 Summary

In this paper, we first detailed the data sources and the various transformations required to prepare these datasets for model training. Subsequently, we introduced the deep neural networks employed in this study, along with the benchmarking methods and mathematical techniques integrated into our research.

The results indicate that the developed system is generally effective at recognizing traffic signs in images captured by cameras mounted on cars. However, its performance in frame-by-frame predictions is limited, primarily due to the insufficient size of the training dataset.

To address this limitation, we adapted an existing algorithm to improve the network's predictions. This approach incorporates a temporal aspect, allowing the system to retain predictions from the previous ten seconds and track them within a unified coordinate system. By employing this method, we achieved accurate three-dimensional predictions, enabling us to determine when a vehicle approaches a speed limit sign and to provide event-based predictions accordingly.

Future work will involve further testing and optimization, particularly concerning the size of the training and validation datasets. With larger datasets available, I believe this system could be effectively integrated into a real-world automotive application.

References

- [1] Gary Bradski and Adrian Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, 2008. URL: <https://www.bogotobogo.com/cplusplus/files/0Reilly%20Learning%20OpenCV.pdf>.
- [2] DataCamp. YOLO Object Detection Explained: A Comprehensive Guide. <https://www.datacamp.com/blog/yolo-object-detection-explained>.
- [3] W. A. Heiskanen and H. Moritz. “The earth-centered, earth-fixed coordinate system and its application in geodesy”. In: Physical Geodesy 111 (1967).
- [4] H. W. Kuhn. “The Hungarian Method for the Assignment Problem”. In: Naval Research Logistics Quarterly 2 (1955), pp. 83–97. DOI: 10.1002/nav.3800020109. URL: <https://doi.org/10.1002/nav.3800020109>.
- [5] Sándor Kunsági-Máté et al. “Accurate Automatic 3D Annotation of Traffic Lights and Signs for Autonomous Driving”. In: (). URL: <https://arxiv.org/pdf/2409.12620>.
- [6] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: arXiv preprint arXiv:1405.0312 (2014).
- [7] Tamas Matuszka et al. “aiMotive Dataset: A Multimodal Dataset for Robust Autonomous Driving with Long-Range Perception”. In: International Conference on Learning Representations 2023. URL: <https://openreview.net/forum?id=LW3bRLlY-SA>.
- [8] Tamás Matuszka et al. “aiMotive Dataset: A Multimodal Dataset for Robust Autonomous Driving with Long-Range Perception”. In: (2022). DOI: 10.48550/ARXIV.2211.09445. URL: <https://arxiv.org/abs/2211.09445>.
- [9] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: arXiv preprint arXiv:1506.02640 (2016). URL: <https://arxiv.org/abs/1506.02640>.
- [10] Roboflow Universe. Self-driving Cars Dataset. 2023. URL: <https://universe.roboflow.com/selfdriving-car-qtyw/self-driving-cars-lfjou/dataset/6>.

-
- [11] Zhi Tian et al. “Fully Convolutional One-Stage 3D Object Detection on LiDAR Range Images”. In: arXiv preprint arXiv:2205.13764 (2022). Accepted to: Proc. Thirty-sixth Conference on Neural Information Processing Systems (NeurIPS) 2022. URL: <https://doi.org/10.48550/arXiv.2205.13764>.
- [12] Ultralytics. YOLOv GitHub Repository. <https://github.com/ultralytics/ultralytics>. 2023.