German Railway Signaling Detection System

Bruno Guiomar

Faculty of Engineering and Computer Science
Department Computer Science,
Hamburg University of Applied Sciences,
Hamburg, Germany
Bruno.Guiomar@HAW-Hamburg.de.

Abstract

Railway safety has evolved considerably over the years, due to constant research and development of supervision systems that analyze driving behaviors and take action when needed. For these systems to be aware of rail safety aspects, it is essential to recognize all track-affected signals. This paper focuses on real-time detection of the German Railway Signaling System, using different YOLOv8 models trained with a custom dataset and the respective results compared in order to establish a conclusion on different real-life applications of this object detection system. To simplify the user experience and conceal the intricacies of the system, a dashboard was developed, aiming to provide an effortless and intuitive interface for users.

Keywords: German Railway Signaling, Real Time Object Detection, Machine Learning, Image Augmentation, YOLO, Python

1 Introduction

In recent years, the fusion of computer vision and artificial intelligence has been revolutionizing the industry and computer science fields. Object detection plays an important role in many real-life applications, ranging from autonomous driving, anomaly identification, video surveillance, and medical imaging.

When dissecting the field of autonomous driving, there is the task of perceiving the vehicle's surroundings, which can then trigger an action or warning. This is one of the most critical and sensitive parts of any autonomous driving system because, a false or wrong detection could cause an accident or inefficiencies. It is important to keep in mind that these systems should only be implemented when they surely have a positive impact on the security of the vehicle and its surrounding environment.

Following this idea, on the railway field, some systems aim to impose safety by continuously analyzing the driver behavior and actions taken in response to some rail-affected signals, but they rely on physical objects placed on the tracks for each

individual signal. This method brings high reliability but a huge cost, considering that the complexity of railway signaling demands a high number of signals for each track. Therefore, they focus only on the most critical signals, which is not ideal.

In this paper, a new method for the German Railway Signal System detection is proposed using machine learning and real-time object detection that can scale up both reliability and safety features while reducing costs and improving overall railway safety.

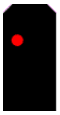
2 German Railway Signal System

Originally, the railway company of each German state had its own signaling system, which raised multiple difficulties when trains began to be able to travel between the different states. After these companies were merged into the German Imperial Railway (Deutsche Reichsbahn), a common signaling system, the H/V system [1], was created based on two key types of signals.

The H/V (Hauptsignal/Vorsignal) [1] system, which consists of main signals and distant signals, is the basis of railway circulation and the one that requires the driver's utmost attention. Main signals are capable of indicating "Proceed" (**Hp 1**, Fig. 1a), "Proceed at reduced speed" (**Hp 2**, Fig. 1b), or "Stop" (**Hp 0**, Fig. 1c). Distant signals replicate the aspect of the upcoming main signal. They are placed at a braking distance before the main signal so that the driver can act in advance and, if necessary, bring the train to a stop before the main signal.

(a) Hp 1 - "Proceed"

(b) Hp 2 - "Proceed at reduced speed"



(c) Hp 0 - "Stop"

Fig. 1: Hauptsignale

(a) Vr 1 - "Expect proceed"

(b) Vr 2 - "Expect proceed at reduced speed"

(c) Vr 0 - "Expect stop"

Fig. 2: Vorsignale

With the H/V system [1], the driver knows when to proceed and stop, however, a crucial piece of information related to each specific section of the track is the speed limit. This is displayed by a number (Fig. 3) that when multiplied by 10, gives the speed limit for that section of the track.

Fig. 3: Lf 7 (Geschwindigkeitssignal) - 120 km/h Speed Limit

There are many more types of signals and different systems in Germany that can represent multiple indications and provide information about the tracks [2], but in this paper, the focus remains on the Deutsche Bahn Hauptbahn (Main Line) signaling system [3].

3 Dataset

The basis for a successful machine learning model is, among others, the dataset. In the context of computer vision and object detection, it should contain representative, accurate, and consistent data so that the training process can build a successful model. Building a good dataset can be both challenging and time-consuming. However, it is essentially an iterative process that consists of experimenting, analyzing training results, inference metrics and refining, in order to correct errors and achieve better results at each version.

3.1 Image Extraction

The core of the German Railway Signaling dataset [4] is composed of images extracted from real-life videos captured by a camera placed in multiple positions inside and outside the train cab.

The type of train or service (passenger or cargo) used for the recordings is not important since the main objective is to capture as many signals as possible on each route. Given that the speed of the train has a direct impact on the quality of image extraction from the video, while higher speeds result in a lower quality image and lower speeds result in a better quality image, both services have their own advantages and disadvantages. Passenger services capture station signals better, since they reduce their speed on the approach, but they perform worse when capturing inter-station signals. This occurs because they typically achieve greater speeds (up to 160 km/h or above for ICE trains), depending on the individual line section. Cargo services usually capture inter-station signals better, due to their lower speed (up to 100 or 120 km/h), but they perform worse when capturing station signals because they don't usually stop.

The image extraction process was made with a Python script (Code List. 1) that takes a screenshot upon detecting a keypress.

```
import keyboard as k
import time
import os
from PIL import ImageGrab
while True:
    if k.is_pressed("p"):
        SS = ImageGrab.grab()
        save_path = os.getcwd() + "/" + str(time.time()) + ".png"
        SS.save(save_path)
```

Code Listing 1: Screenshot Python Script

3.2 Image Augmentation

Achieving a well-represented dataset across all classes isn't an easy task, especially in the signaling field. Similarly to other signaling systems, some signals are more common than others, and that also happens in the railway context.

The H/V system [1] signals (Fig. 1, 2) are the most frequent in the German railway, followed by the three main signal distance indicators (Ne 3 class, Fig. 6e, 6f, 6g) and speed limit signals (Fig. 3), which are also very common. On the other hand, there are rarer signals that can only be found on specific sections of the route or in some specific conditions. For the first scenario, Ne 5 signals (Fig. 6c) can only be seen on stations since their purpose is solely to indicate the different stooping positions for each train length. The second scenario can be illustrated with dynamic signals, which can change their aspect at any time, specifically semaphores that represent the aspect of main signals or distant signals. Some aspects like Hp 2 (Fig. 1b) or Vr 2 (Fig. 2b) aren't as common as their Hp 1 "Proceed" (Fig. 1a) and Vr 1 "Expect Proceed"(Fig. 2a) siblings.

In order to balance class representation on the dataset, the classes with unbalanced lower representation were first pre-augmented. Then, to achieve better data representation and variety, the complete dataset was augmented. This general augmentation led to an increase in the number of images from 1605 to 3877 in the final version 6. For this, a simple Python script (Code List. 2), implemented with the Augmentor [5] library, was created using the following parameters described on Table 1:

```
import Augmentor

p = Augmentor.Pipeline("hp0_img")
p.random_brightness(probability=0.8, min_factor=0.33, max_factor=2)
p.random_contrast(probability=0.9, min_factor=0.3, max_factor=2.1)
p.zoom_random(probability=0.45, percentage_area=0.70, randomise_percentage_area=
                                        False)
p.shear(probability=0.55, max_shear_left=9, max_shear_right=9)
p.sample(30)
```

Code Listing 2: Augmentation Python Script

Type	Probability (%)	Min	Max
Brightness	0.8	0.33	0.2
Contrast	0.9	0.3	2.1
Zoom	0.45		
Shear	0.55		9

Table 1: Augmentation Parameters

Brightness (Fig. 4b) and **Contrast** (Fig. 4c) helps to simulate different lightning conditions and camera settings.

Zoom (Fig. 4d) simulates multiple signal distances.

Shear (Fig. 4e) tries to represent the train inclination on curves.

(a) Original Image

(b) Brightness Parameter

(c) Contrast Parameter

(d) Zoom Parameter

(e) Shear Parameter

Fig. 4: Augmentation Example

Figure 5 shows the overall balance between all dataset classes before the augmentation process. This relative balance was achieved through class pre-augmentation. In previous versions of the dataset, poorly achieved results were due to heavily unbalanced classes, so this is an important step to keep in mind.

Fig. 5: Overall Class Balance

3.3 Dataset Overview

Signal Classes: The Deutsche Bahn Hauptbahn (Main Line) signaling system [3] contains numerous signals and variations representing complex scenarios and properties of the tracks. Building an object detection working model for the complete system would be a demanding and possible task, but it does not comply with the objectives of this project. To achieve a simple working model, the most relevant signals were chosen. Besides the H/V system [1] signals (Fig. 1, 2), figure 6 shows the other dataset signal classes.

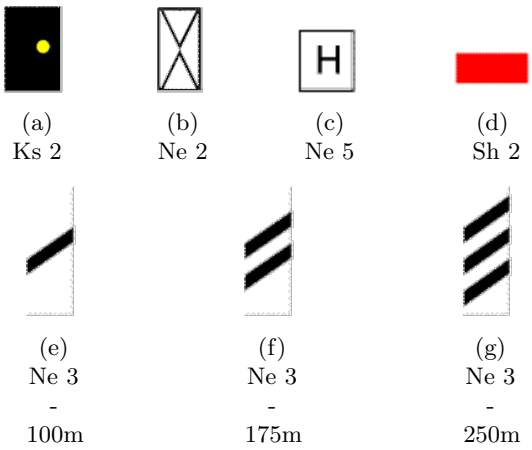


Fig. 6: Dataset Additional Classes

In order to provide basic knowledge about the meaning of these signals, a brief explanation is given. **Ks 2** (Fig. 6a) belongs to the KS system, which is a family of signals that can represent both main and distant signals, depending on the respective post plate below them. In this case, KS 2 can represent "Line clear" in the context of main signal and "Expect stop" in the context of distant signal. **Ne 2** (Fig. 6b) is a post plate signal that is placed below all distant signals. **Ne 5** (Fig. 6c) is placed exclusively on stations and marks the train stop positions. **Sh 2** (Fig. 6d) is one of the most important signals because it represents the end of the line. **Ne 3 Family** (Fig. 6e, 6f, 6g) signals indicate the respective distance to a main signal.

Global Metrics: Table 2 shows the dataset global metrics after the augmentation process.

Images	Annotations	Classes
3877	4937	13

Table 2: Dataset Global Metrics

Class Average Metrics: Table 3 shows the average image distribution per class before the augmentation process. Initially, the images were divided by groups (train, valid, and test) using a popular formula where 70% of the images stay in the training group, 20% are separated in the valid group, and the remaining 10% stay in the testing group. In order to achieve a more refined balance, individual images were strategically relocated among groups.

Image Group	Range	Average
Train	101-124	112.15
Valid	27-36	30.46
Test	13-18	14.62

Table 3: Class Image Division

4 Object Detection

After a robust dataset is built, the object detection model must be created so it can be used by the inference engine. This model is the core of any object detection system, and its complexity and architecture have a direct influence on inference speed and precision. In this project, different YOLOv8 [6] pre-trained models were chosen as a training basis due to their speed and precision.

4.1 Training

The training process consists of feeding the dataset through a model architecture and adjusting its parameters on each iteration to improve performance. This process can be both challenging when choosing and tuning the right training configurations and extremely time-consuming, especially for larger datasets.

In order to take advantage of the significant advancements and good results of the YOLOv8 pre-trained object detection models, three of these models (Table 4) were custom trained with the project's dataset.

Model	mAP[1]	Speed (ms) CPU ONNX	Speed (ms) A100 TensorRT	Params (M)
YOLOv8n	37.3	80.4	0.99	3.2
YOLOv8s	44.9	128.4	1.20	11.2
YOLOv8m	50.2	234.7	1.83	25.9

Table 4: YOLOv8 Pre-trained Detection Models reference data [7]

YOLOv8n is the most lightweight model of the object detection family, and it counts with an average accuracy (compared with the other models) expressed by the average mAP value and a notorious inference speed due to the low number of parameters. This model is most suitable for low-performance devices, even with CPU inference processing.

YOLOv8s has an accuracy improvement compared to the previous nano model, but with the increase in parameters, it also reduces performance. It is suitable for mobile deployments.

YOLOv8m is the medium model in the object detection family, and it has a substantial increase in the number of parameters, which negatively impacts inference time while slightly increasing accuracy.

Training a custom model requires a large amount of computing power, and specific types of hardware perform better on these operations, like Graphics Processing Units (GPUs), Tensor Processing Units (TPUs) together with a good amount of RAM (Random Access Memory). Since these pieces of hardware are usually very expensive and are not available to everyone, there are some cloud computing services that provide a remote environment with specific hardware that is essential for training custom models. For this project, Google Colaboratory [8] was used to provide a cloud environment with a GPU and RAM where the training process was performed.

The custom training configuration is made easy using the YOLOv8 API [9]. With a simple command (Code List. 3), the training process can be configured and initiated.

[1] Values are for single-model single-scale on COCO val2017 dataset.

```
yolo task=detect mode=train model=<model_name> data=<dataset_location> epochs=<
                                        epoch_number> imgsz=<image_size> plots=
                                        True
```

Code Listing 3: YOLOv8 Training Command

Training Configuration: all three models were trained with the same configuration to allow an easy result comparison. An image size of 640px and 150 epochs.

4.2 Inference Engine

After the custom model is built, it can now be used to detect objects in new, unseen data as input. In object detection, the input data is usually an image that goes first through a pre-processing phase where scaling and optimizations are made, followed by the concrete inference phase where image data is fed to the model, and finally the results are processed. The output data of an object detection inference can be in the form of a list, where classes that were detected are listed together with their respective confidence values, or the original image with the detected classes bounding boxes overlapped. These two output options have varying impacts on inference time due to differences in their processing complexities, with the latter being slower.

Model complexity can directly affect inference time, but the processing hardware used can also dramatically change this important variable. As seen in Table 4, using TPUs to process inference calculations has a substantially positive impact on inference time when compared with CPU processing, so they are preferable both for training and inference stages.

Some changes can be made to the input image in order to optimize inference performance. In the context of this project, the input image is vertically split by approximately half its width and extracted only from its right half. This procedure has two advantages. First, the standard practice in the German railway system typically involves using the right track for normal circulation, resulting in the affected signal appearing on the right side of the track. This can be proved by analyzing the Annotation Heatmap graphic (Fig. 7), where the average location of the annotations is marked as green. Second, this split lowers the image size by half, reducing considerably the number of processed pixels, which, in this case, lowers the inference time. This conclusion is not general and cannot be taken as granted for all applications, so it is important to analyze if the time taken to split the image is lower than the saved time on the inference process.

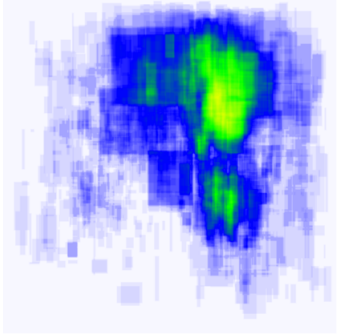


Fig. 7: Dataset Annotation Heatmap

The inference engine was wrapped in a Python class that performs all three required steps. **Image Extraction**, **Inference**, and **Results Processing**. Code Listing 4 shows a simplified version of the inference method.

```python
from ultralytics import YOLO

def inference(model_path, frame):
    model = YOLO(model_path)
    # Feed YOLOv8 model with frame
    results = self.model(frame)
```

Code Listing 4: Inference Method

Image Extraction Method is responsible for collecting the input data and preprocessing it. It includes the image split method and conversion to a numpy array.

Inference Method is responsible for feeding the custom model with the output from the previous method and storing the results.

Results Processing Method processes the raw output, arranges and structures the information, generates a frame with overlapping annotations, and then updates the graphical user interface (Sec. 5).

5 Dashboard

Graphical user interfaces (GUIs) can be a great transparency tool for any kind of software. While improving visualization and control, GUIs also make the software usable for any kind of user (even without advanced context knowledge). For this project, a simple GUI was developed in Python with the PyQT6 [10] framework, and it is both a visualization and control tool (Fig. 8).

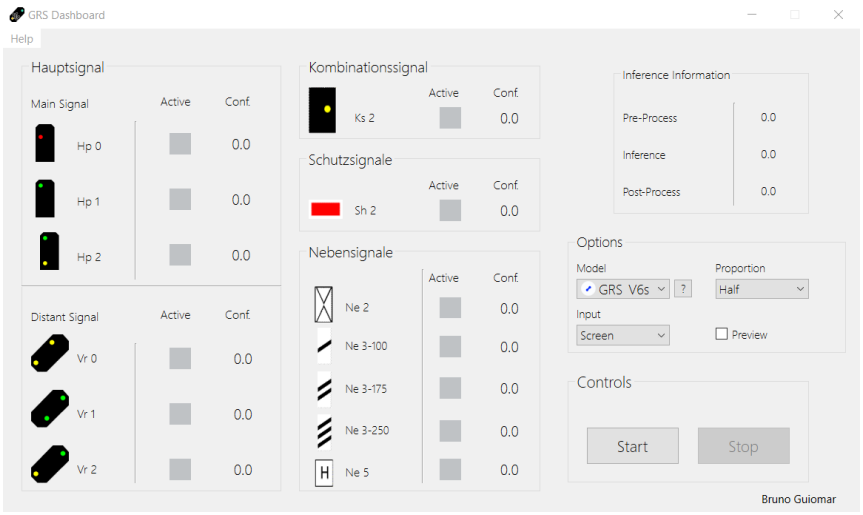


Fig. 8: Dashboard

The dashboard can be divided into four sections. **Signal Detection View** with its respective confidence value, **Inference Information** including real-time metrics related to the different inference stages, **Options** where the model can be chosen or its individual information accessed, screen proportion, inference input source, and preview window toggle, where a pop-up window can be created with the detected

classes annotation boxes overlapping the input source, and finally the **Controls** where the inference engine can be started or stopped.

6 Results

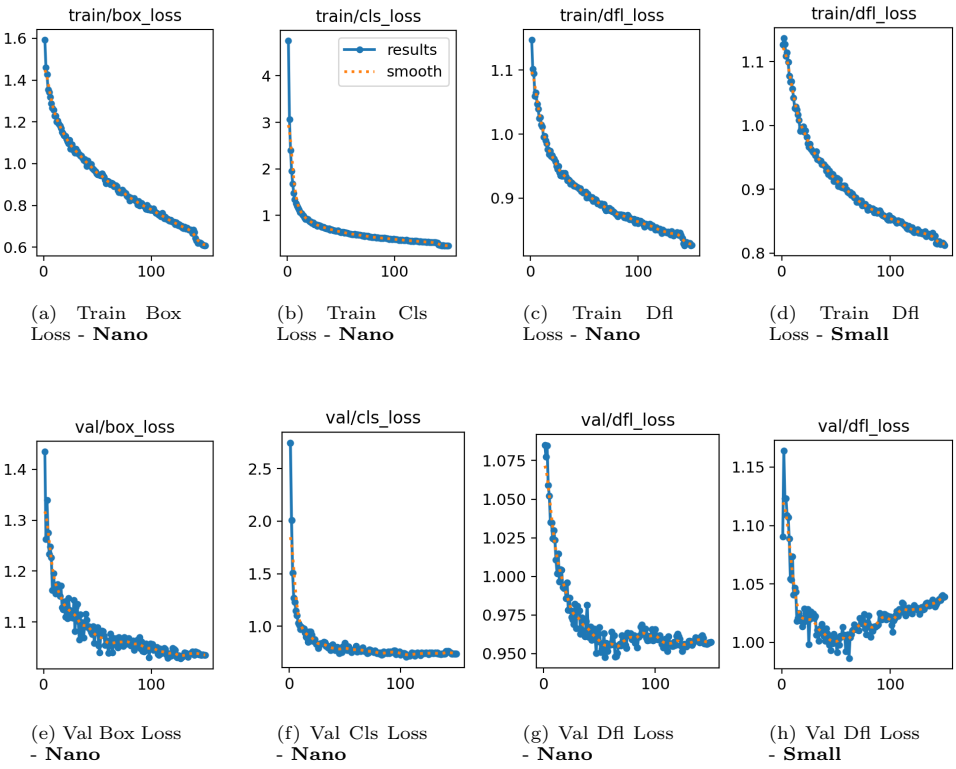
In this section, the results from the different stages of the project are shown and explained.

With the aim of achieving a model with the best properties, three different models were custom trained with the same configurations but based on distinct YOLOv8 object detection architectures explained in section 4.1. Table 5 establishes a comparison among the training results of the different models.

Architecture	Precision	Recall	mAP
YOLOv8n	83.5%	84%	86.8%
YOLOv8s	86.3%	85.9%	89%
YOLOv8m	86.3%	85.9%	89%

Table 5: Training Model Metrics Comparison

Understanding the model behavior during the training process can lead to important conclusions and problem diagnoses. A close analysis of this information can also indicate the model's success and imperfections, which can unravel minor issues that should be resolved. Figure 9 contains training and validation loss graphics from the custom trained model based on YOLOv8 nano and small architectures. These graphics represent how well the model is performing during training, relating the epoch number on the X-axis and Loss value on the Y-axis. The loss value metric has an inverse relationship with the model's performance, so lower values are preferable.



(a) Train Box Loss - **Nano**

(b) Train Cls Loss - **Nano**

(c) Train Dfl Loss - **Nano**

(d) Train Dfl Loss - **Small**

(e) Val Box Loss - **Nano**

(f) Val Cls Loss - **Nano**

(g) Val Dfl Loss - **Nano**

(h) Val Dfl Loss - **Small**

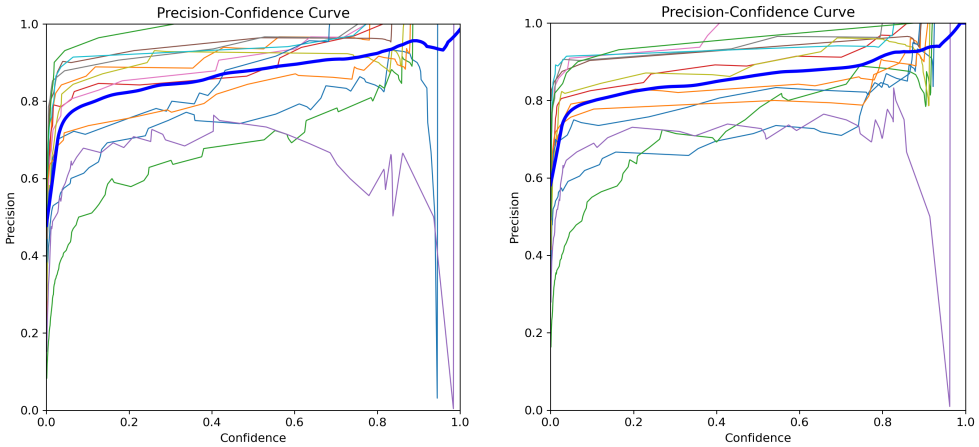
Fig. 9: Loss Graphics for Nano and Small Architecture models

The **Box Loss metric (box_loss)** reflects the model's performance concerning bounding box coordinates and dimensions. Analyzing figure 9a, it is possible to conclude that the training process was successful due to the descending curve, but it may have ended prematurely since this curve didn't reach a stable point, closer to a horizontal line. The Validation graphic (Fig. 9e) also shows a descending curve, closer to stability, which validates the model's learning process when tested with different images.

The **Classification metric (cls_loss)** represents the model's ability to correctly classify predictions. Figure 9b shows an early descending curve with a stable tendency at the end, which also shows that the learning process was effective. The early descent suggests that the model achieved a fast and remarkable classification ability, and its stability indicates that a longer training process won't have a strong influence on this metric. The validation graphic (Fig. 9f) also supports these conclusions.

Finally, the **Distribution Focal Loss [11] (dfl_loss)** expresses errors in the deformable convolution layers, which influences the model's ability to handle object deformations and appearance variations. Both nano and small models share a similar curve when analyzing their train graphics (Fig. 9c, 9d). These similarities end when comparing the two validation graphics (Fig. 9g, 9h), with the nano model's curve first decreasing with a little ascent followed by relative stability and overall oscillations. This behavior is not a cause for significant concern, given its steep descent and relative stability. When compared with the small model's curve (Fig. 9h), after the normal descent, there is a concerning climb in the loss values. This implies that the model's performance within the DFL context was progressively deteriorating over time.

Object detection model performance can be represented using multiple metrics with different meanings. The **Precision-Confidence Curve** graphic establishes a relation between the model's ability to avoid detecting false positives and the detection confidence rate. Analyzing its curve can lead to critical conclusions that should be taken into account when deploying the model. Figure 10 shows a comparison between Precision-Confidence Curve graphics for the project's custom-trained models based on YOLOv8 nano (Fig. 10a) and small (Fig. 10b) architectures. Each class is distinguished by a uniquely colored line identified in the legend on Figure 10c, where the dark blue line represents the average curve.



(a) Precision-Confidence Curve - **Nano**

(b) Precision-Confidence Curve - **Small**

(c) Class Legend

Fig. 10: Precision-Confidence Curve Comparison

The **Precision-Confidence** graphic's general curve (dark blue line) indicates that at lower confidence rates, the model is not able to avoid false positives, which means that some false positive detections can occur with a low confidence rate. Usually, this does not represent a problem because, normally, lower confidence rate detections are ignored. A general rising curve is favorable as it signifies that when detections have higher confidence rates, the model detects fewer false positives and gains the ability to avoid them. The challenge emerges in particular classes where, despite a confidence rate nearing 1.0, there is a substantial drop in precision values. This can be a big problem because it means that the model can detect false positives with a high confidence rate in those classes and not ignore them. When comparing the nano and small graphics, it becomes apparent that this problem exists in both, although less pronounced in the small model.

With the previous graphic, it is possible to establish an abstract quantification of the model correctness and false positive occurrence, but after this general analysis, it is important to determine which specific classes are the false positives occurring, how often and which classes are being incorrectly identified as others. The **Confusion Matrix** is a table that can identify specific classification problems in a model per

class. Figure 11 shows the Confusion Matrix for the custom-trained model based on the YOLOv8 nano architecture.

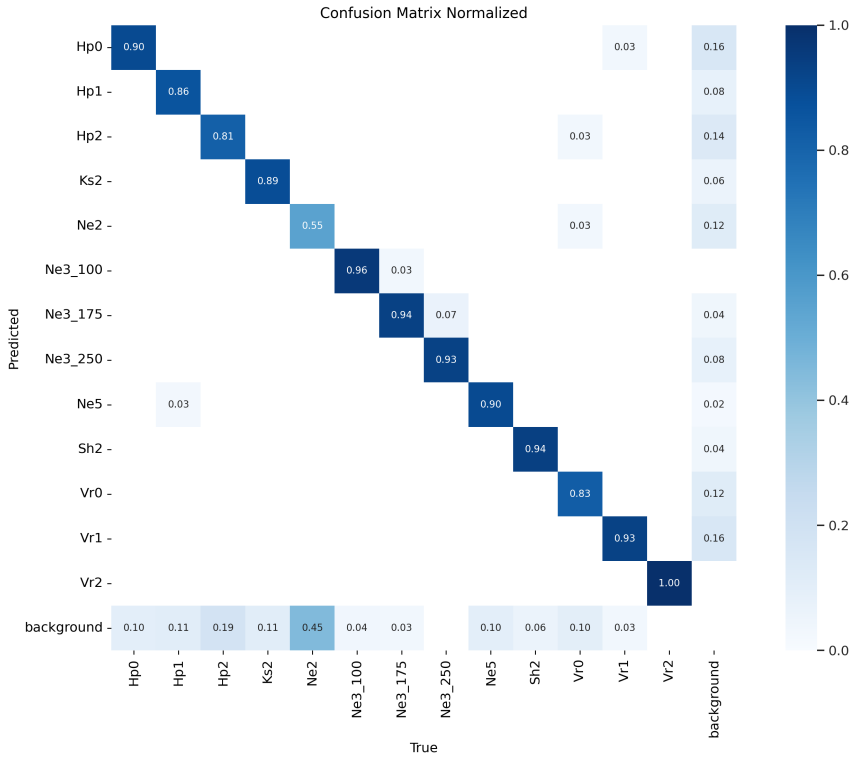


Fig. 11: Confusion Matrix Normalized - **Nano**

When analyzing the nano model normalized **Confusion Matrix**, it is possible to confirm the model's successful classification learning process due to its high correctness, having every class correctly classified above 50%, only one class below 80% and one class with 100% correctness. The main failure point, according to this confusion matrix, is false detections in the background (detecting objects when there are any).

The final phase of an Object Detection project is the deployment process, where the created model is fed with a non-dataset image and, if the previous phases were successful, the objects are correctly detected with an appropriate bounding box and classified with a high confidence rate. Achieving a model as perfect as described earlier is a challenging and iterative task of trial, error, and correction, so regularly deploying and testing the model is a step to success. Figure 12 shows inference examples of the custom-trained model based on the nano architecture in both real-life and simulation scenarios.

(a) Real Life Image (b) Train Simulator Image

Fig. 12: Inference Example Comparison

7 Real Life Applications

Object detection systems have a wide range of applications in the real world. They can be a trigger for different actions, mass object classification, or even detecting anomalies in different contexts. This project emerged from the identification of two practical, real-world applications that can use railway signaling detection. **Autonomous train driving** (Sec. 7.1) and **Train Driving Assistance** (Sec. 7.2).

In both cases, there is a common advantage. Signals are physical objects placed along train tracks, and they were not originally thought to be detected and interpreted by computers, so the majority of the signals can't be directly detected by a board computer placed on a train.

In Germany, to address this issue, a safety system called PZB (Punktförmige Zugbeeinflussung / Punctiform tension control) [12] was created, whose objective was to make certain signals and aspects detectable by the board computer on each train and the expected action from the driver supervised. If the driver didn't act as expected according to the signal indication, the train automatically activates the emergency brakes, coming to a stop. The problem with this system is that for a signal to be detected, a physical magnet (Gleismagnet) should be placed on the track next to each signal. Since it is not feasible to place these magnets on the tracks for every single signal on the German railway, engineers came up with the solution to just focus on the most critical signals and aspects.

The incredible advantage of using the Object Detection system is that there's no need to change the rail infrastructure in order to reach the same objective as PZB [12] and even go further by detecting all signals in the rail network.

7.1 Autonomous Driving

Having a computer controlling big machines is not an easy task, mostly due to safety issues. Defining when a system is "ready" for being in charge of a powerful machine and human lives can be both a philosophical and technical challenge, but certainly before even thinking about that scenario, the system should manifest a level of advanced robustness and success.

The system developed for this project is a simple demonstration of machine learning and computer vision capabilities when executing tasks like detecting signals, but it is clearly not ready for independently driving a train without human interaction. False or incorrect detections can lead to unnecessary train stops or rail accidents.

7.2 Driving Assistance

This Object Detection System (ODS) can have a positive impact when combined with the existing safety systems that follow the concept of assisting the train driver.

Following this concept, the detection system can be perceived as an additional feature of PZB [12], but with less authority due to its lower confidence rate. Since PZB [12] only covers a small group of signals (H/V system [1] (Fig. 1, 2) and some speed change aspects), the rest of the signals can be covered by the ODS as assistance to drivers.

The majority of speed limit change signals are not detected by the PZB system [12], and they only appear once in a small Lf 7 sign like this (Fig. 3), so it is easy for the train driver to miss this sign or to forget its value. With ODS, when the Lf 7 sign (Fig. 3) or any other speed limit change signal is detected, it can automatically record its value, show it in the train's graphical interface, and impose an internal speed limit on the train system.

An additional prospective feature involves establishing a backup system for PZB [12] in the event of a failure or implementing a secondary detection mechanism to prevent false conclusions and detect errors effectively.

Rail accidents can be difficult to investigate, particularly when determining if it's a matter of human error. The signal detection system can provide additional information to the train event recorders, enabling the construction of a precise sequence of events, crucial for determining the root cause of the accident.

8 Conclusion

This project has demonstrated that railway signaling detection using a custom-trained model based on YOLO's architecture is possible and successful.

Three different architectures were trained with the custom dataset, and the overall best model is the one based on YOLO's **nano** architecture, due to its high precision and light weight structure. It is suitable for most use cases with low-power computation hardware and can achieve the lowest inference time. The other two architectures, small and medium, achieve the exact same results when trained with the same dataset and configurations but have overall high precision and confidence rates. Both models also require high computation power and have a higher inference latency attached to them.

In conclusion, designing and implementing a successful Object Detection system depends on the efficacy of its various stages. Each stage involves a process of experimentation, analysis, and correction, comprising numerous iterations, until a previously defined acceptable state is reached. The performance of each phase is intricately linked to the preceding one, underscoring the significance of every step in the process.

9 Prospects

This project can undergo further enhancements and incorporate new features. This section presents ideas aimed at realizing potential improvements.

Dataset expansion with more types of signals and different images from uncovered regions. A new class should only be included if there is enough data to represent it, in order to maintain class balance. This rule also applies when adding new data to existing classes.

Implementing a **self-expanding module**. After the system is deployed on multiple locomotives, each node not only performs inference but also captures images of both low- and high-confidence detections and automatically sends the collected data to a central database where it can be stored and verified. This module can exponentially increase dataset representativeness and quickly map all rail network signals. This module's output shouldn't skip accurate verification or replace precise manual data collection and labeling.

Inference engine adaptation for compact processing units. Trains are equipped with diverse hardware from various systems, making size a crucial factor in this context. The inference process should be carried out and optimized for a compact processing unit. This task involves a hardware accelerator and robust model optimization for the specific hardware used.

References

[1] JMRI: DB HV System 1969. https://www.jmri.org/xml/signals/DB-HV-1969/index.shtml. Accessed: 23/02/2024

[2] Wolfgang Meyenberg: German Railway Signal Systems. https://www.sh1.org/eisenbahn/s.htm. Accessed: 24/02/2024

[3] DB Netze: Richtlinien 301 – Signalbuch, Aktualisierung 11, 11 edn. Mainzer Landstraße 185, 6ß327 Frankfurt am Main (2020). DB Netze

[4] Bruno Guiomar: German Railway Signaling Dataset. https://universe.roboflow.com/haw-hamburg-g6n2v/german-railway-signaling-dataset. Accessed: 19/02/2024 (2023)

[5] Bloice, M.D., Stocker, C., Holzinger, A.: Augmentor: An Image Augmentation Library for Machine Learning (2017)

[6] Jocher, C.A.Q.J. Glenn: Ultralytics YOLO. Accessed: 22/02/2024 (2023). https://github.com/ultralytics/ultralytics

[7] Ultralytics: YOLOv8 Object Detection. https://docs.ultralytics.com/tasks/detect/. Accessed: 21/02/2024 (2023)

[8] Google: Google Colaboratory. https://colab.research.google.com/. Accessed: 21/02/2024 (2017)

[9] AyushExel, Glenn-jocher, Laughing-q: YOLOv8 API. https://docs.ultralytics.com/pt/reference/cfg/__init__/. Accessed: 23/02/2024 (2023)

[10] Riverbank Computing: PyQt. https://www.riverbankcomputing.com/software/pyqt/. Accessed: 20/02/2024 (2021)

[11] Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., Tang, J., Yang, J.: Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection (2020)

[12] Brigitte Stodtmeister: Punktförmige Zugbeeinflussungsanlagen Bedienen Auf Fahrzeugen Mit LZB-Einrichtungen. Deutschen Bahn AG, (2009). Deutschen Bahn AG