

MASTER THESIS

Thesis submitted in fulfillment of the requirements for the degree
of Master of Science in Engineering at the University of Applied
Sciences Technikum Wien - Degree Program Data Science

Multi-sensor rail track detection in automatic train operations

By: Attila Kovacs

Student Number: 2110854031

Supervisors: Lukas Rohatsch, MSc

Dr. Daniele Capriotti

Wien, May 17, 2024

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsge-
setz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, May 17, 2024

Signature

Kurzfassung

Die Erhöhung des Automatisierungsgrades im Bahnbetrieb ist ein Schlüsselfaktor für ein effizienteres und wettbewerbsfähigeres Eisenbahnsystem. Automatische Zugbetriebssysteme (ATO) nutzen fortschrittliche Technologien, um die Eisenbahnumgebung wahrzunehmen und zu interpretieren und um autonome Betriebsabläufe mit minimalem menschlichem Eingriff zu ermöglichen. Die präzise Identifizierung und Lokalisierung von Eisenbahngleisen sind grundlegend für eine sichere Zugnavigation. Daher besteht Bedarf an robusten automatisierten Methoden zur Gleiserkennung. Diese Masterarbeit befasst sich mit der Aufgabe der Mehrsensorschienenverfolgung im Rahmen von ATO durch die Nutzung von Deep-Learning-Techniken, insbesondere der semantischen Segmentierung. Durch die Analyse eines Mehrsensordatensatzes, der Bilder von hoch-/niedrigauflösenden RGB-Kameras und Infrarotkameras mit unterschiedlichen Orientierungen umfasst, zielt diese Arbeit darauf ab die Genauigkeit und Robustheit der Gleiserkennung unter verschiedenen Bedingungen zu verbessern. Die Studie vergleicht die Leistung der Deep-Learning-basierten Segmentierung mit traditionellen nicht-KI-basierten Methoden, führt umfassende Bewertungen verschiedener Sensorinputs durch und untersucht die Integration in realen Anwendungen. Die Ergebnisse liefern wertvolle Erkenntnisse auf dem Gebiet der Eisenbahnautomatisierung und bieten Implikationen zur Verbesserung der Sicherheit und Effizienz des automatischen Zugbetriebs.

Schlagworte: Deep Learning, Computer Vision, Rail Track Segmentation, YOLO, Automatic Train Operations

Abstract

Increasing the level of automation in train operations is an enabler for a more efficient and competitive railway system. Automatic train operations (ATO) systems use advanced technologies to perceive and interpret the railway environment in order to facilitate autonomous operations with minimal human intervention. Precise identification and localization of railway tracks are fundamental for safe train navigation. Hence, there is a need for robust automated track detection methods. This master's thesis addresses the task of multi-sensor rail track detection within the ATO framework by utilizing deep learning techniques, particularly semantic segmentation. By exploring a multi-sensor dataset comprising images from high/low-resolution RGB cameras and infrared cameras with varying orientations, this research aims to improve the accuracy and robustness of track detection in diverse conditions. The study compares the performance of deep learning-based segmentation with traditional non-AI-based methods, conducts comprehensive evaluations on different sensor inputs, and explores real-world application integration. The findings contribute valuable insights to the field of railway automation, offering implications for enhancing the safety and efficiency of automatic train operations.

Keywords: Deep Learning, Computer Vision, Rail Track Segmentation, YOLO, Automatic Train Operations

Acknowledgements

The author would like to thank Lukas Rohatsch and Daniele Capriotti for their support and insightful contributions. Acknowledgment is also extended to DB and DB Cargo for providing the essential images and videos that made this research possible. The computational infrastructure was graciously provided by the UAS Technikum Wien.

Contents

1	Introduction	1
2	Literature review	2
2.1	Traditional rail track detection	3
2.2	Deep learning-based rail track detection	4
2.3	Lane detection	5
3	Datasets	5
3.1	OSDaR23 dataset	6
3.1.1	Overview	6
3.1.2	Brightness of the images	9
3.1.3	Entropy of the images	10
3.1.4	Occlusion	11
3.1.5	Images and video frames	13
3.2	RailSem19 dataset	13
3.3	Data splitting	15
4	Solution approach and experiments	16
4.1	Infrastructure	17
4.2	Transforming labels	18
4.3	Performance evaluation	20
4.4	Non-AI-based segmentation	22
4.4.1	Fast line detection	23
4.4.2	Postprocessing	23
4.4.3	Parameter tuning	25
4.5	Deep learning-based segmentation	27
4.5.1	Models	27
4.5.2	Hyperparameter tuning	31
5	Results	32
5.1	Performance analysis	32

5.2 Post-processing	39
5.3 Integration into real-world applications	39
6 Conclusion	41
Bibliography	43
List of Figures	50
List of Tables	53
A Appendix	54
A.1 Sensors used in OSDaR23	54
A.2 Dataset split	54
A.3 Fast line detection	56
A.4 YOLOv8	57
A.4.1 Results of data_highres_railsem_640 model with unified labels . .	58
A.4.2 Results of data_all_640 model with unified labels	59
A.4.3 Results of data_highres_railsem_640 model with separate labels .	60
A.4.4 Results of data_all_railsem_640 model with separate labels . . .	61
A.4.5 Parameter tuning with unified labels	62

1 Introduction

According to the International Energy Agency (IEA), the global demand for passenger and freight transportation will more than double by 2050 compared to 2019 (International Energy Agency, 2019). However, a greater demand entails higher energy consumption as well as increased CO₂ emissions and atmospheric pollutants. Given the fact that railway is one of the most efficient and reliable modes of transportation, there seems to be a consensus between politicians and researchers that a greater share of rail transportation has the potential to counterbalance the negative impacts of transportation (Islam et al., 2016; Pagand et al., 2020). The IEA lists minimizing costs per passenger-kilometer or ton-kilometer moved as one of three pillars that are essential to increase the market share of rail transportation¹.

Automatic Train Operations (ATO), which refers to a system that automates different aspects of train operations is expected to be one of the key drivers of a more efficient and competitive railway system (ALSTOM Transport SA, 2021; Europe's Rail Joint Undertaking, 2019). ATO is estimated to reduce energy consumption by up to 45%, increase the level of punctuality, increase operational flexibility, and allow for a 50% better utilization of the infrastructure when combined with other technologies.

ATO relies on advanced technologies that are used to perceive and interpret the railway environment in order to allow autonomous operations with minimal or no human intervention (Deutsche Bahn AG, 2022). One aspect of ATO is the precise identification and localization of railway tracks. The ability to detect and isolate tracks based on video images is essential for ensuring the safe navigation of trains through the railway network or in shunting yards. Accurate track detection ensures that the train can make informed decisions, such as adjusting speed, navigating turns, and responding to potential obstacles.

Traditional methods of track detection often rely on rule-based algorithms and image processing techniques but these approaches may face challenges in diverse environmental conditions such as bad weather, complex background, lighting variations (e.g.,

¹The other pillars are maximizing revenues from rail systems and ensuring that all forms of transport (especially road transportation) pay not only for the use of the infrastructure they need, but also for the adverse impacts they generate.

day and night), and dirty cameras. This master's thesis addresses the task of multi-sensor rail track detection in the context of ATO. We explore deep learning techniques (particularly semantic segmentation) that have demonstrated great success in computer vision tasks. The application of deep learning to track detection is expected to outperform conventional non-AI-based techniques and thereby improving the accuracy and robustness of the system. Our analysis is based on a multi-sensor dataset, including images of regular RGB cameras, high-resolution cameras, and infrared cameras with different orientations, respectively. This multi-sensor approach allows us to compare the effectiveness of different camera types and informs the deployment of those in order to improve the robustness of track detection in diverse conditions.

In the context of rail track detection, researchers have explored various areas. Yet, applying deep learning techniques to detect rail tracks is a relatively raw field. In particular, there is no research that is focusing on comparing different input images such as RGB and infrared cameras and images that are oriented to the left, center, and right of the locomotive. The contribution of this thesis to the literature is three-fold: First, we select and train different deep learning models capable of accurately detecting and segmenting railway tracks using data from RGB cameras, high-resolution cameras, and infrared cameras. In contrast to approaches that have been specifically tailored to the task, we apply a general framework that is easier to use by practitioners without elaborate software engineering skills. The results of the deep learning model are compared to a non-AI-based method specialized in identifying lines in images. Second, we conduct a comprehensive performance evaluation to assess the accuracy and computational efficiency of the proposed track detection system on images generated by different cameras. Third, we explore the integration of the developed tool into real-world applications by applying the model to identify tracks in video streams. The framework includes a post-processing step that incorporates explicit domain knowledge in order to enhance the robustness of the deep learning results.

By achieving these objectives, this research provides valuable insights and advancements to the field of railway automation with implications for improving the safety and efficiency of automatic train operations.

2 Literature review

Traditionally, rail track detection has been performed by first extracting features of an image (e.g., gradient-based thresholds) and then detecting rails. These approaches achieve good results in certain conditions. However, deep learning-based approaches are often more robust in real-world environments (Giben et al., 2015; Li and Peng, 2022; Wang et al., 2019). Deep learning techniques, particularly convolutional neural networks (CNNs), have emerged as powerful tools for image segmentation tasks, demonstrating success in various computer vision applications. Recent surveys on image segmentation and object detection using deep learning techniques are provided by Cheng et al. (2023) and Zaidi et al. (2022), respectively.

The following sections examine related research in track detection which consider both deep learning-based segmentation and traditional non-AI segmentation methods.

2.1 Traditional rail track detection

While deep learning has shown remarkable success in track detection, non-AI segmentation techniques continue to play a role in this field as they allow the integration of domain-specific knowledge and rules into the algorithm and require less data for training. These methods are often referred to as line segment detectors and involve traditional computer vision techniques such as thresholding, edge/contour detection, template matching, and region growing (Almazàn et al., 2017; Grompone von Gioi et al., 2010, 2012; Mittal et al., 2022; Sahoo et al., 1988).

Kaleli and Akgul (2009) present a dynamic programming algorithm to extract the rail tracks in front of the train. The idea is to first identify the vanishing point which refers to the imaginary intersection of the tracks as the distance between the tracks decreases from the bottom of the image to the top. This step is based on computing the gradient and applying Hough transform to detect the straight lines that indicate the tracks. Next, dynamic programming is used to extract the space between the two tracks. Qi et al. (2013) apply a method based on histogram of oriented gradients (HOG) to identify tracks and switches. First, HOG features are computed; railway tracks are then identified by a region-growing algorithm. The proposed method is able to predict the path the train will travel by detecting the setting of the switches. Nassu and Ukai (2011) introduce an approach that performs rail extraction by matching edge features to candidate templates.

While the previously mentioned approaches focus on images by on-board cameras, Purica et al. (2017) examine the detection of tracks in aerial images taken by drones. The solution approach is based on Hough transform.

Arastounia (2015) and Yang and Fang (2014) develop methods to recognize railroad infrastructure from 3D LIDAR data. In Arastounia (2015), railway components such as rail tracks, contact cables, catenary cables, masts, and cantilevers are classified based on local neighborhood structure, shape of objects, and topological relationships among objects. Yang and Fang (2014) focus on the detection of tracks. The authors utilize the geometry and reflection intensity of the tracks to extract features and identify tracks.

2.2 Deep learning-based rail track detection

Deep learning-based techniques incorporate CNNs and other deep architectures to automatically learn features from raw image data. Semantic segmentation in particular assigns a label to each pixel in the image, distinguishing between the pixels that belong to the rail tracks and those that represent the background. The approach is therefore well suited for rail track detection.

Giben et al. (2015) and Le Saux et al. (2018) were among the first authors who evaluated the performance of deep learning-based segmentation against traditional segmentation techniques in rail track detection. In Giben et al. (2015), the authors propose a CNN for localizing and inspecting the condition of railway components based on gray-scale images. The authors report that the CNN model is better suited to capture complex patterns compared to approaches that rely on traditional texture features (e.g., discrete Fourier transforms of local binary pattern histograms). Le Saux et al. (2018) detect rail tracks in aerial images by devising a CNN based approach and different traditional approaches such as thresholding.

Wang et al. (2019) propose the RailNet – a deep learning-based rail track segmentation algorithm that combines the ResNet50 backbone with a fully convolutional network. In order to train the model, the authors compile a non-public dataset consisting of 3000 images from forward-facing on-board cameras. Experiments show that RailNet is able to outperform general purpose models for segmentation. Li and Peng (2022) compile a real-world railway dataset based on which a rail detection method referred to as Rail-Net is devised. Rail-Net outperforms traditional methods by around 51% and other deep learning methods by around 6% based on accuracy.

A machine learning-based approach is proposed by Teng et al. (2016) where features are extracted from super-pixels (i.e., a group of adjacent pixels with similar characteristics) and classified by applying a previously trained support vector machine.

2.3 Lane detection

Lane detection for road vehicles is similar to rail track detection for locomotives in the sense that both tasks aim to identify and segment elongated shapes in complex environments that vary in lighting conditions, shadows, and occlusions. The field of lane detection has a rich body of literature which is among others attributed to the existence of well-established benchmark datasets such as TuSimple (2017) and CULane (Pan et al., 2018).

Early work in lane detection is based on traditional approaches such as Hough transform and clustering (Duda and Hart, 1972; Ma and Xie, 2010). Recently, the focus of researchers has shifted to deep learning-based approaches (Meyer et al., 2021; Wang et al., 2022; Zheng et al., 2022). Tang et al. (2021) and Yang (2023) provide comprehensive surveys on lane detection approaches. In Yang (2023), the authors propose an approach in which the advantages of traditional and deep learning-based methods are combined.

3 Datasets

Labeled images are an essential prerequisite for training deep learning algorithms to detect objects accurately. With the growing popularity of deep learning, there has been an ongoing effort to create new datasets specifically designed for railway applications.

The rail semantics dataset 2019 (RailSem19) is the first publicly available dataset for detecting objects (including rail tracks) in the railway domain (Zendel et al., 2019). The French railway signaling dataset (FRSign) is a dataset focusing only on traffic lights (Harb et al., 2020), whereas the Railway Pedestrian Dataset (RAWPED) is focusing on pedestrian detection methods (Toprak et al., 2020). The dataset proposed by Wang et al. (2019) – railroad segmentation dataset – has been compiled for the development of railroad segmentation algorithms but it is not available to the public. The Rail-DB

dataset comprises 7.432 annotated images, featuring different scenarios (e.g., weather conditions), and is available upon request (Li and Peng, 2022).

This thesis is based on the first freely available multi-sensor dataset “Open Sensor Data for Rail 2023” (OSDaR23) for the development of fully automated driving in the railway sector (Deutsche Bahn AG, 2023; Tagiew et al., 2023). Unlike the previously mentioned datasets that involve a limited number of sensors and perspectives, the system on the locomotive used to create the OSDaR23 dataset includes multiple infrared cameras, RGB cameras with different resolutions, lidar, radar, positioning, and acceleration sensors.

Preliminary experiments indicated that our segmentation model fails to generalize when trained only on the OSDaR23 dataset due to reasons that will be described in the next section. Therefore, we also train our model on images from the RailSem19 dataset. In the following, we give a detailed description of the two datasets used in this thesis.

3.1 OSDaR23 dataset

In this section, we introduce the OSDaR23 dataset in more detail and explore the characteristic of the used images in terms of labeling, brightness and entropy.

3.1.1 Overview

The OSDaR23 contains 21 video sequences captured around Hamburg, Germany between 09.09.2021 and 15.09.2021 (a map of the exact locations is given in Figure 1).

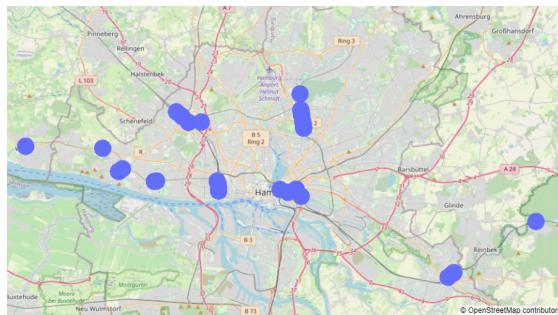


Figure 1: Locations where images were captured around Hamburg, Germany.

The sensor setup includes six RGB cameras, three infrared cameras, six lidar sensors, a 2D radar sensor, and position and acceleration sensors. In this thesis, we focus on

images by RGB high resolution, RGB low resolution, and infrared sensors with three orientations (left, right, and center), respectively. One example per sensor is given in Figure 2. A detailed description of the sensors can be found in Appendix A.1.

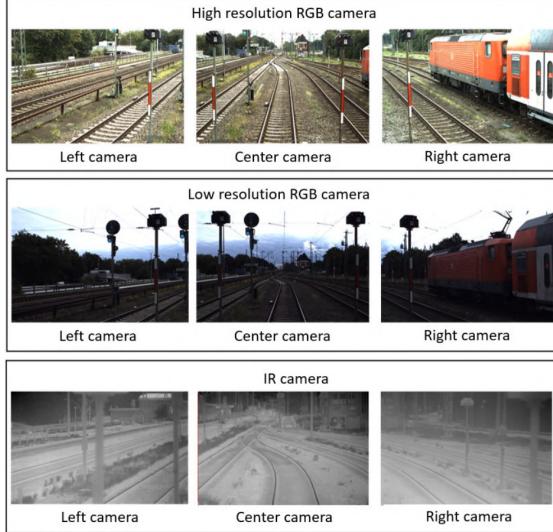
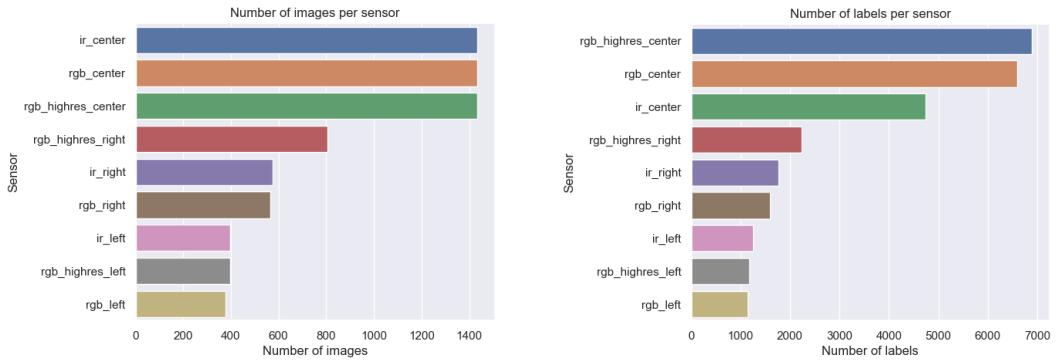


Figure 2: Example images of high resolution RGB, low resolution RGB, and infrared sensors (Tagiew et al., 2023).

The final number of images and labels after filtering the dataset, i.e., removing images that do not contain annotated tracks, is 7.421 and 27.386, respectively. The distribution of images and labels per sensor is displayed in Figure 3. The size of the images is given in Table 1. Figure 4 illustrates the number of track labels per image. Most images contain track pairs. However, there are also images with an odd number of tracks. The largest group are images containing one pair of tracks. Generally, the number of available images decreases as the rail network is getting more complicated.



(a) Number of images per sensor.

(b) Number of labels per sensor.

Figure 3: Number of images and labels per sensor, respectively.

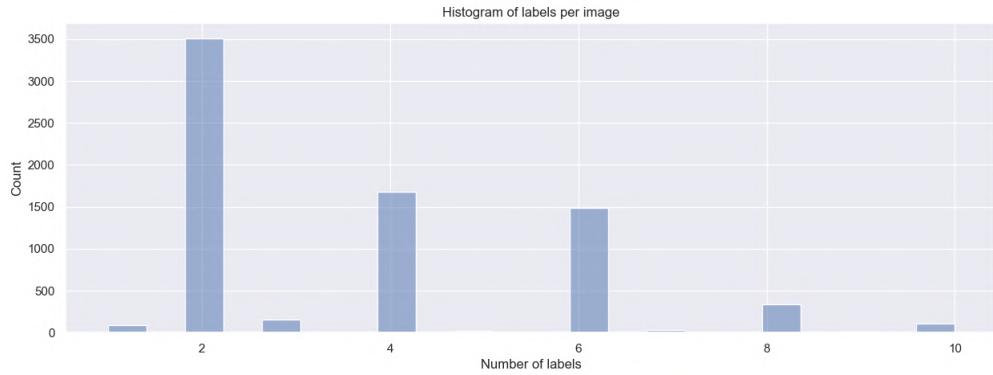


Figure 4: Track labels per image. Most images depict track pairs. However, there are also images with an odd number of tracks.

Sensor	Width [px]	Height [px]	Aspect ratio
RGB low resolution	4112	2504	1.64
RGB high resolution	2464	1600	1.54
Infrared	640	480	1.33

Table 1: Size of images per sensor.

All images were taken between 8AM and 5PM, so we cannot expect to test the effect of different sensors in the night. In particular, the RGB cameras fail to capture clear and detailed images in low-light conditions. Infrared cameras on the other hand, detect infrared radiation emitted by objects based on their temperature rather than visible light.

and are used in low-light conditions or complete darkness¹. The thermal radiation is converted into electrical signals which are then processed to a visual image that is visible to the human eye (Clark et al., 2002). Warmer areas appear as brighter shades of gray, while cooler areas appear darker.

Emissivity, a material property that indicates how efficiently an object emits infrared radiation, plays a significant role in thermal imaging. Emissivity is measured on a scale from 0 to 1, where 0 indicates a perfect reflection of the radiation (no emission such as a mirror), and 1 indicates perfect emissivity (total emission in an object referred to as blackbody). Detecting rail tracks in infrared images is based on the principle that polished metallic surfaces such as tracks have a low emissivity, whereas organic materials that appear often in the background have a high emissivity.

3.1.2 Brightness of the images

In deep learning, the quality of the images can have a large impact on the efficiency. In image segmentation tasks that have the goal to identify and classify each pixel in an image are particularly sensitive to variations in pixel brightness and intensity. In this section we analyze the brightness of the images for each type of sensor. Brightness is defined as the average pixel intensity $\frac{1}{N} \sum_{i=1}^N I_i$ (N is the number of pixels in the image and I_i is the intensity of pixel i). Figure 5 shows a series of box plots with brightness values per sensor.

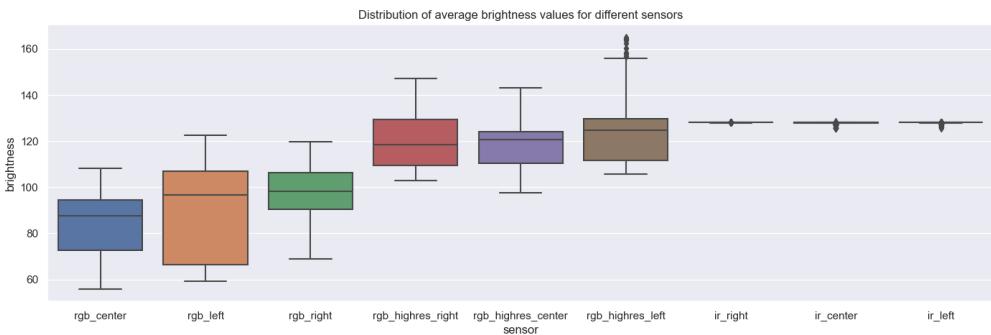


Figure 5: Brightness of images by sensor.

Among the three types of sensors, low resolution RGB cameras produce the darkest images (a black image has a value of 0, a white image has a value of 255). High resolution images are brighter on average, which can be explained by different exposure

¹All objects with a temperature greater than absolute zero emit infrared energy.

settings such as shutter speed and ISO sensitivity. Both low and high-resolution cameras feature pixels of equal size ($3.45\mu m$), so the amount of light per pixel is the same. Infrared cameras produce images with almost constant brightness as the non-visible infrared image is mapped on a visible spectrum.

Figure 6 shows one example of a bright and a dark image, respectively.

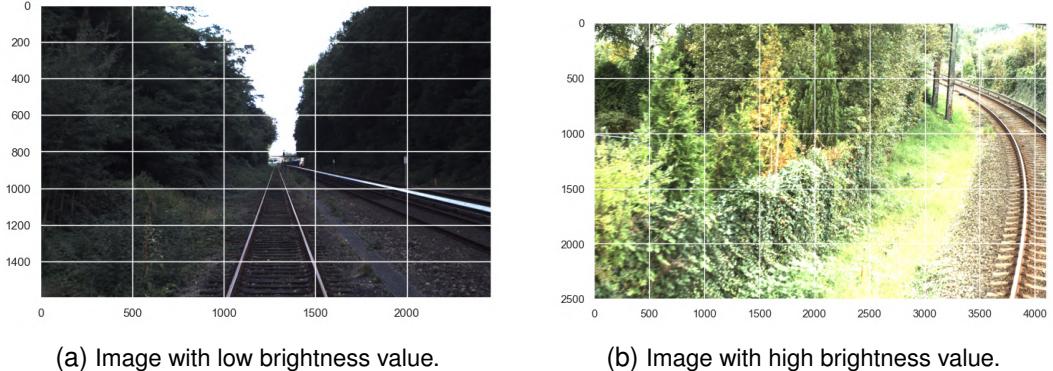


Figure 6: Examples of very bright and very dark images, respectively.

3.1.3 Entropy of the images

Shannon entropy is a measure from information theory that reflects the uncertainty or randomness associated with a set of data (Shannon, 1948). In the context of images, Shannon entropy can be used to quantify the complexity in the pixel values of an image. A high entropy value indicates higher complexity or randomness in the pixel values, while a low entropy value suggests more homogeneity. Rahane and Subramanian (2020) report a positive correlation between the entropy of the training data and the performance of semantic segmentation tasks, highlighting that more complex images are harder to learn by deep learning networks. The Shannon entropy for a grayscale image is given by $H(X) = - \sum_{i=1}^n P(x_i) \cdot \log_2(P(x_i))$, where $P(x_i)$ is the probability of occurrence of pixel x_i (i.e., the number of pixels with intensity x_i divided by the total number of pixels). A box-plot with the entropy distribution is given in Figure 7 for each sensor. The maximum entropy value is $8 = \log_2(256)$ as we convert the images to grayscale with 256 different intensity levels.

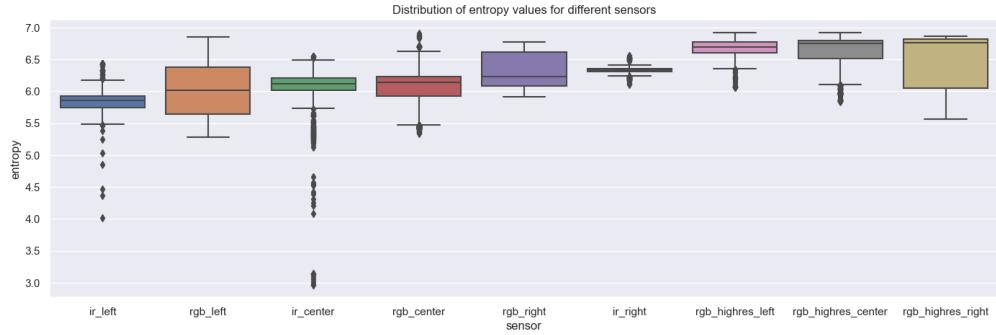
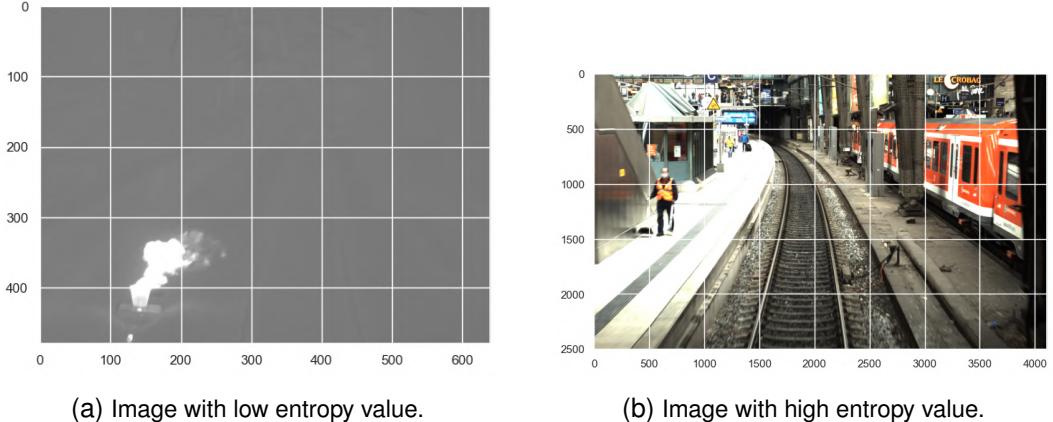


Figure 7: Entropy of images by sensor.

Overall, the images have a high entropy – the median values range from 5.9 to 6.8. High resolution images have the highest entropy. Infrared images do not seem to have lower entropy on average. However, certain images have a very low randomness, and as it turns out, also a very low level of information when looking at Figure 8a. Figure 8 highlights the visual difference between a very low and a very high entropy image.



(a) Image with low entropy value.

(b) Image with high entropy value.

Figure 8: Examples of images with minimal and maximal entropy, respectively.

3.1.4 Occlusion

Certain track labels are hidden or occluded by other objects. One example is given in Figure 8b where the train on the right covers the tracks. In this section, we analyze the occlusion of the labels and examine those occlusions visually.

Figure 9 shows the number of labels with a given occlusion level. Most of the labels, 20.069, are not occluded at all or have only a slight occlusion. However, 320 labels in

196 images are marked with an occlusion level of 100%. Figure 10 shows two examples where the track labels are fully covered.

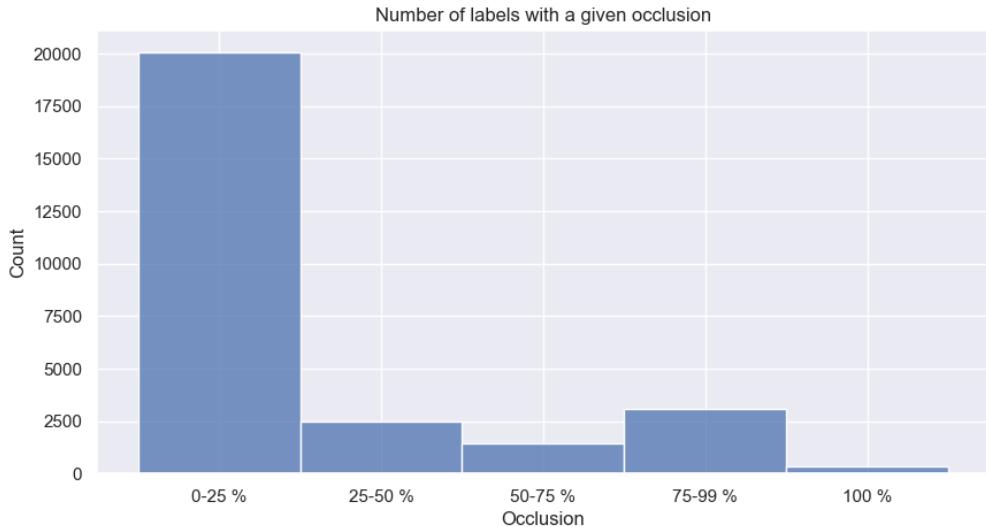
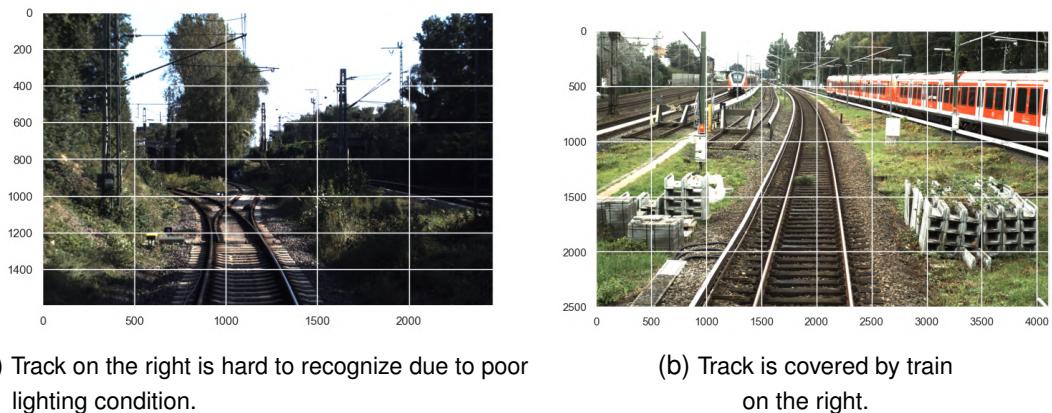


Figure 9: Histogram showing the occlusion level for track labels.

We keep all images in the dataset as the CNN might recognize that there has to be a track below a train and because most images with covered labels contain visible labels as well.



(a) Track on the right is hard to recognize due to poor lighting condition.

(b) Track is covered by train on the right.

Figure 10: Examples of track labels with 100% occlusion.

3.1.5 Images and video frames

Rail tracking systems aim to analyze video streams by treating each video frame as an independent image. Consequently, the images in the OSDaR23 dataset represent individual frames extracted from video sequences. Figure 11 shows three examples of video sequences with seven frames each. It is clear that there is only minor variation in the images.



Figure 11: Three examples with seven video frames (i.e., images), respectively.

Two major issues, when training a model on a dataset that consists of similar images, are a lack of generalization (i.e., inability to generalize well on diverse and unseen images) and reduced robustness (i.e., vulnerability to variations in lighting conditions and backgrounds). In order to mitigate the issues, we add images from the RailSem19 dataset to the training data.

3.2 RailSem19 dataset

The RailSem19 dataset (Zendel et al., 2019) is not the primary focus of this thesis. However, the previous analysis reveals that among the 7.421 images within the OSDaR23 dataset, a significant number show high similarity. This similarity is due to the fact that the images are frames from a video sequence and the presence of three cameras for each orientation. The RailSem19 dataset is added to the training set in order to increase the performance of the segmentation approach. In the following, we will examine the RailSem19 dataset in more detail.

The dataset consists of 8500 rail images taken in different countries, and weather and lighting conditions. The number of rail annotations is 58.483. All images have a size of

1920x1080 pixels. Figure 12 shows a histogram of images with the respective number of track labels on a log scale. Most images contain 4 track labels (i.e., two pairs of tracks) but the dataset also contains complicated networks with 26 pairs of tracks. An example of a simple and an example of a complicated infrastructure is given in Figure 13.

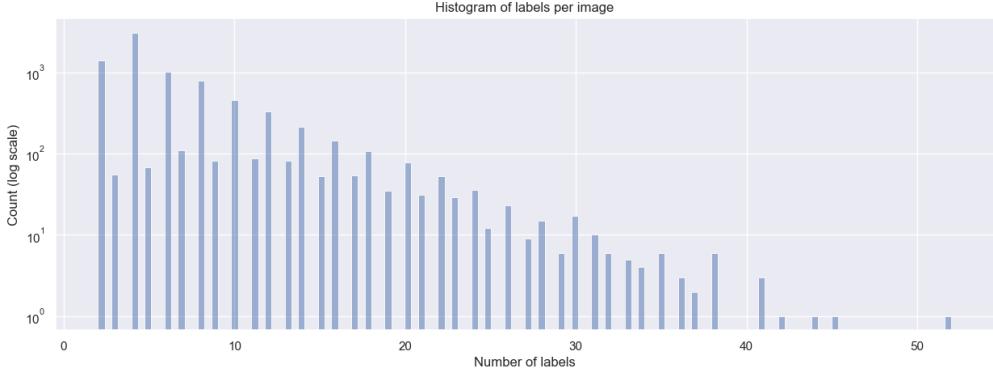


Figure 12: Track labels per image on a logarithmic scale. The images range from simple railroads with a single pair of tracks to complicated networks with 26 pairs of tracks.

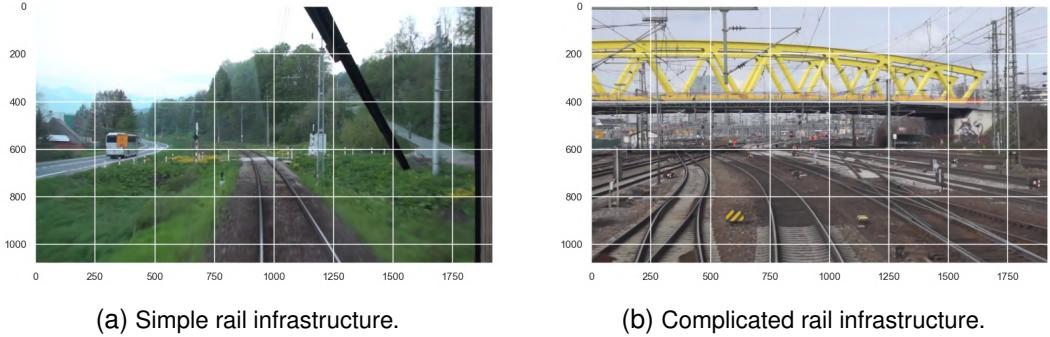


Figure 13: Examples of simple and complicated rail infrastructures.

Figure 14a and Figure 14b show the distribution of the brightness values and the entropy values as defined in Section 3.1, respectively. The RailSem19 images are mostly brighter than the OSDaR23; the distribution of the values ranges from almost black images to almost white images (see Figure 15a and Figure 15b). The entropy of the RailSem19 images is similar to the images in the OSDaR23 dataset. The outliers at the lower end of the entropy spectrum are associated with infrared images in the OSDaR23 dataset. However, in RailSem19 low entropy images are often images that were generated in tunnels or at night. Figure 15b and Figure 15d are images with low

entropy levels close to zero whereas Figure 15c has the highest entropy level in the data set with a score of 6.97.

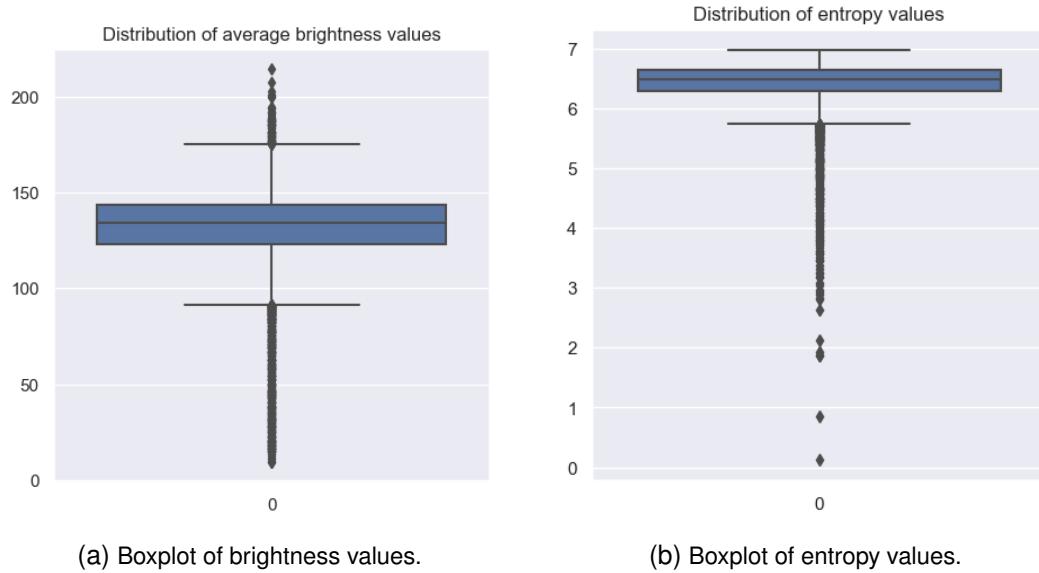


Figure 14: Brightness and entropy of RailSem19 images.

3.3 Data splitting

Splitting a dataset into training, validation, and test sets is a standard practice in machine learning to evaluate and improve the performance of a model. Separating data into distinct sets and using them at different stages of development prevents unintended data leakage where information from the test or validation set influences the training process.

The training set is used to train the model – the model learns patterns and relationships within the training data. The validation set is used to evaluate the performance during the training process, fine-tune the hyperparameters of the model, and to prevent overfitting (i.e., a situation when a model performs well on the training data but fails to generalize to new data). Adjustments to the algorithm without compromising the model's ability to generalize to new data can be made by evaluating the performance on the validation set. The test set is reserved for the final evaluation of the model. The test set allows for an unbiased assessment of the model's performance on data it has never seen before and provides, therefore, an estimate of how well the model is likely to perform in real-world applications.

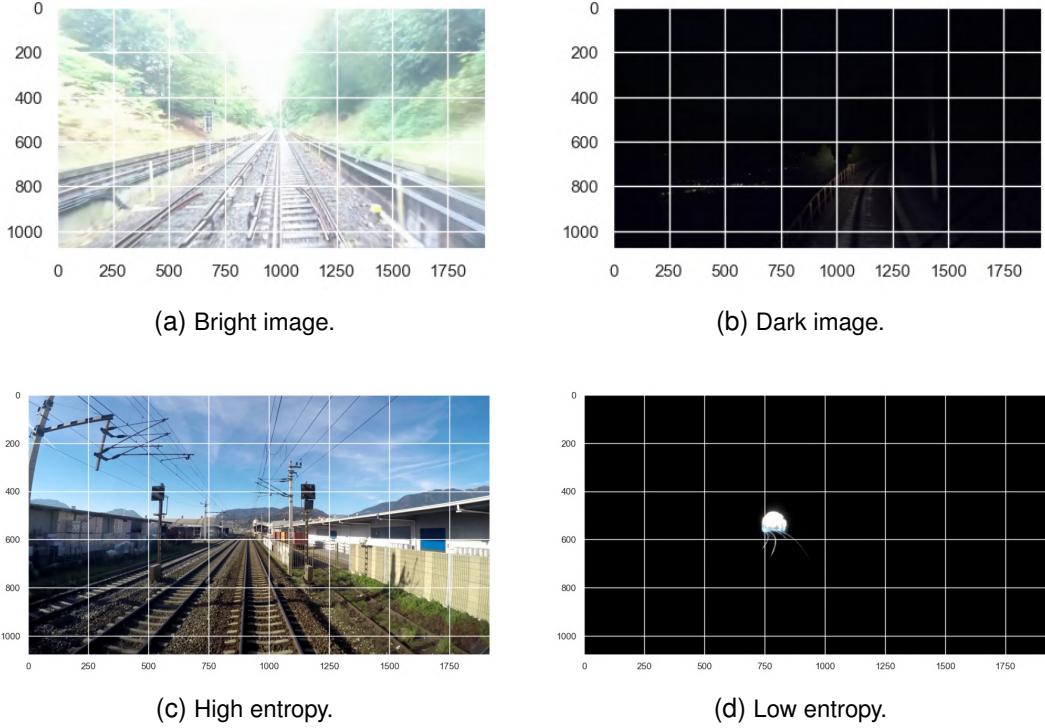


Figure 15: Extreme examples with regard to brightness and entropy.

We perform a random split with a share of 70%/15%/15% for the train, validation, and test sets, respectively. Splitting the RailSem19 set is straightforward; we randomly assign images to each set resulting in 5.950, 1.275, and 1.275 images per set, respectively. However, the OSDaR23 comprises many images that are very similar (e.g., frames from a video that were taken when the locomotive was standing still), so we perform a random split not on individual images but on video sequences. The approach results in 5.506 images for training, 987 images for validation, and 928 images for testing. The assignment of video sequences to the respective set and the split on sensor level is given in Appendix A.2.

4 Solution approach and experiments

Our objective is to precisely segment the visible rail tracks captured by an on-board camera on the locomotive.

Within computer vision, semantic segmentation is a well-studied domain that focuses on classifying each pixel in an image into predefined categories (Mo et al., 2022). Conceptually, it can be seen as an extension of object detection, a technique that recognizes and precisely locates objects within an image using bounding boxes. The pixel-level precision in semantic segmentation enables the understanding of the image context and is therefore often used in applications like autonomous driving and robotics.

The segmentation is performed by the YOLOv8 algorithm (Jocher et al., 2023). YOLOv8, short for "You Only Look Once, version 8," is the latest version in the YOLO series of real-time object detectors, offering state-of-the-art performance in terms of accuracy and speed (Ultralytics Inc., 2023). YOLOv8 builds upon the YOLO architecture, introduced in Redmon et al. (2016) – a paper that has received more than 43.000 citations at the time of writing this thesis. Since the introduction of the original YOLO in 2016, the framework has been tested extensively with remarkable results (e.g., Diwan et al. (2023); Feng et al. (2022); Jiang et al. (2022)). The efficiency of the YOLO implementations by Jocher et al. (2023) for segmentation tasks is examined in Yue et al. (2023) and Straker et al. (2023).

Meyer et al. (2021) introduce YOLinO, a versatile and efficient algorithm inspired by YOLO. It is designed to detect polylines in the context of lane detection, as discussed in Section 2.3. While the proposed approach is suitable for rail track detection, adapting the existing code base from lane detection to rail track detection exceeds the scope of this thesis. Furthermore, our goal is to employ an off-the-shelf solution framework that can be readily utilized by a broader audience.

In order to evaluate the YOLOv8 approach, we establish a baseline by devising a traditional track detection algorithm based on the fast line detector (FLD) by OpenCV (2024b). In the following, we describe the YOLOv8 and the FLD approach, and give details of the infrastructure for running the experiments.

4.1 Infrastructure

Experiments are run on the High Performance Lab Cluster (HPL) of the UAS Technikum Wien. The HPL comprises 19 PCs connected through a 10-GBit-Network. The PCs

have the following setup, respectively: AMD Ryzen 9 3900X 12-Core Processor, NVIDIA GeForce RTX 2080 Ti GPU, 32 GB RAM, and 80 GB Hard Disk.

All nodes are running Linux Ubuntu 20. CUDA libraries are installed for GPU management. All scripts were written in Python 3.8.10. The code can be accessed upon request at https://github.com/clewfirspockonehanded/rail_detection.

4.2 Transforming labels

In semantic segmentation, labels are often represented as closed polygons. Each polygon corresponds to a specific semantic class and the pixels enclosed within the polygon are assigned the label of that class. However, the track labels in both OSDaR23 and RailSem19 dataset are given as polylines. Therefore, we transform the labels as follows. First, we adjust the thickness of the polyline such that it covers the tracks. Second, we generate the mask – a binary image where each pixel is assigned a value indicating the category it belongs to (i.e., track or background). Finally, the boundaries of the track label are extracted by using the *drawContours* and *approxPolyDP* functions in the OpenCV library. The vectors defining the polygons are written to disk and loaded during the training. Figure 16 illustrates the transformation process.

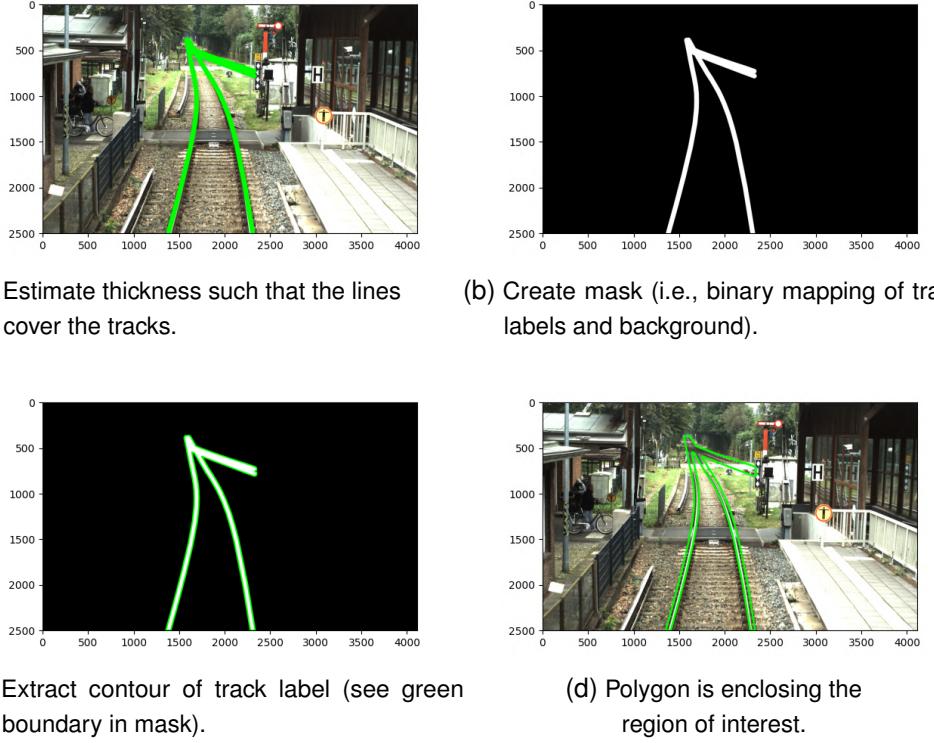


Figure 16: Process of transforming the labels from polyline to polygon.

The presented approach generates polygons based on the segmentation mask. In order to test the effect of the labeling strategy, we also generate labels for each rail track separately. Figure 17 illustrates the difference between the two labeling approaches. The unified polygon based on the mask is shown on the left; on the right, each track is labeled separately.

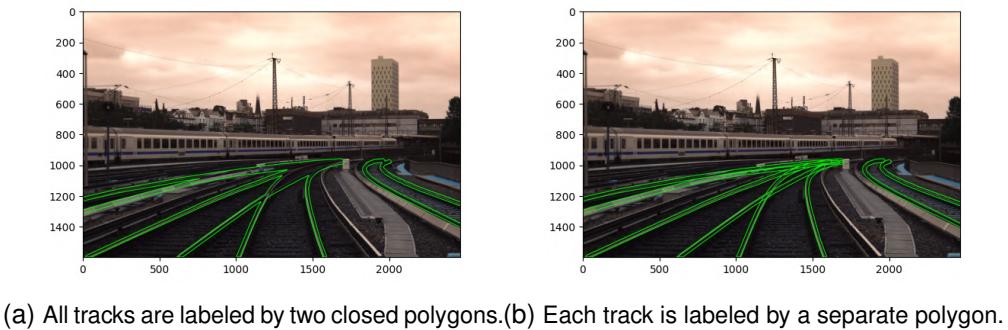


Figure 17: Two different approaches for labeling tracks.

4.3 Performance evaluation

There is a number of performance metrics that are typically used in semantic segmentation to evaluate the effectiveness of the models (Taha and Hanbury, 2015). The choice of metrics depends on the specific requirements of the application and the nature of the dataset. In this section, we will introduce and examine different metrics for the rail segmentation task.

Pixel Accuracy as defined in Equation 1, provides a simple measure of the overall accuracy by considering correctly predicted pixels.

$$\text{Pixel Accuracy} = \frac{\text{Number of Correctly Predicted Pixels}}{\text{Total Number of Pixels}} \quad (1)$$

Precision measures the accuracy of positive predictions among all positive predictions (see Equation 2). The metric is important when the focus is on minimizing false positives (e.g., in medical diagnostics).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

True Positives (TP) are the instances where the model correctly predicts the positive class (i.e., it correctly identifies the track). False Positives (FP) are the instances where the model incorrectly predicts the positive class (i.e., it identifies a track when it is actually absent).

Recall measures the ability to capture all positive instances (see Equation 3). Recall is optimized when it is crucial to identify all instances of a particular class, even at the cost of some false positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

False Negatives are the instances where the model incorrectly predicts the negative class (i.e., it identifies background even when there is a track).

Dice score or F1 score as shown in Equation 4, balances precision and recall which is particularly useful when dealing with unbalanced datasets, i.e., when one class has a much larger or smaller number of pixels compared to the other class (in our case background vs. tracks). Dice and F1 score are the most used metrics in segmentation when the dataset is unbalanced. Some authors use the Dice score while others use the F1 score with the respective formulas given below. However, it can be shown that one reduces to the other (see Equation 5).

$$\text{Dice Score} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \quad (4)$$

$$\begin{aligned} \text{F1 Score} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\ &= \frac{2 \times \frac{\text{TP}}{\text{TP} + \text{FP}} \times \frac{\text{TP}}{\text{TP} + \text{FN}}}{\frac{\text{TP}}{\text{TP} + \text{FP}} + \frac{\text{TP}}{\text{TP} + \text{FN}}} \\ &= \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \end{aligned} \quad (5)$$

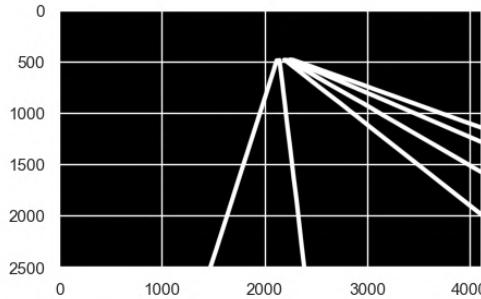
Intersection over Union (IoU) or Jaccard index measures the overlap between predicted and ground truth regions (see Equation 6). The metric is commonly used to evaluate the spatial accuracy of segmentation masks. Apart from the boundary values {0,1}, the IoU score is consistently smaller than the Dice score.

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \quad (6)$$

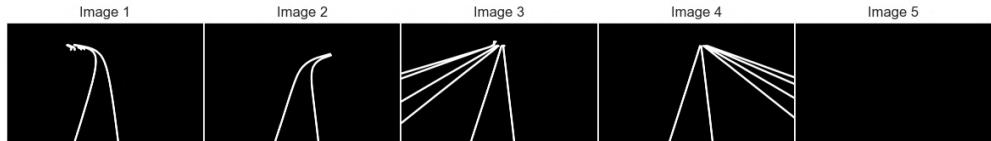
In Figure 18, we illustrate the performance metrics based on an example. Figure 18a shows the ground truth label, i.e., the correct mask of the image being analyzed. Figure 18b shows five possible predictions, where Image 5 is simply a black image with no positive classifications. The metrics for the five predictions are given in Figure 18c. The example shows that Pixel accuracy is not appropriate in unbalanced datasets – even Image 5 which has no prediction at all has an accuracy score of almost 95%. Image 2 yields a high Precision score of 0.706 as the FN values are not accounted for. Recall seems to be a strong predictor of performance in this example. However, an all white image (i.e., all pixels classified as track) would yield a perfect Recall. Dice and IoU are

best suited to distinguish between good and bad predictions. Both values have a wide range and the scores are in line with the estimation based on visual inspection.

Based on the literature and the example, we choose the Dice score to analyze the performance of different models.



(a) Ground truth label selected randomly from training set.



(b) Example predictions selected randomly from training set; last image is only background illustrating a case where no rail track was found.

Image	Pixel Accuracy	Precision	Recall	Dice	IoU
1	0.927	0.013	0.005	0.007	0.004
2	0.956	0.706	0.249	0.369	0.226
3	0.916	0.197	0.201	0.199	0.111
4	0.999	0.992	0.991	0.991	0.983
5	0.948	- ^a	0.0	0.0	0.0

^aDivision by zero as there are no positive classifications.

(c) Performance Metrics.

Figure 18: Example masks – ground truth vs. prediction.

4.4 Non-AI-based segmentation

To assess the performance of the YOLO approach, we establish a baseline using a traditional line detection method. Specifically, we employ the fast line detection algorithm as implemented by OpenCV (2024b). In a post-processing step, we incorporate domain

specific knowledge to improve the results. In this section, we describe the baselining approach in detail.

4.4.1 Fast line detection

The fast line detection algorithm (FLD) as proposed by Lee et al. (2014) operates as follows: First, edges in the image are detected using a Canny edge detector. The system then extracts line segments by connecting edge pixels with straight lines and extending them until they satisfy the co-linearity conditions (i.e., whether or not the newly added pixels align well with the existing segment in terms of forming a straight line). Points that deviate from forming a straight line are considered endpoints of other segments.

FLD is suitable for applications where speed is crucial such as real-time systems or applications where processing resources are limited. The exact implementation details vary based on the specific version of OpenCV and underlying optimizations that are applied.

4.4.2 Postprocessing

The results of the FLD algorithm are post-processed to incorporate domain specific knowledge. More precisely, we filter out lines based on the observation that cameras with different orientations capture rail tracks that are bounded by boxes of different aspect ratios. Figure 19 illustrates the idea: Figure 19a shows an image of a forward facing camera; the aspect ratio of the bounding box enclosing the track is large. Figure 19b shows an image of a left facing camera; the aspect ratio of the bounding box enclosing the track is small.

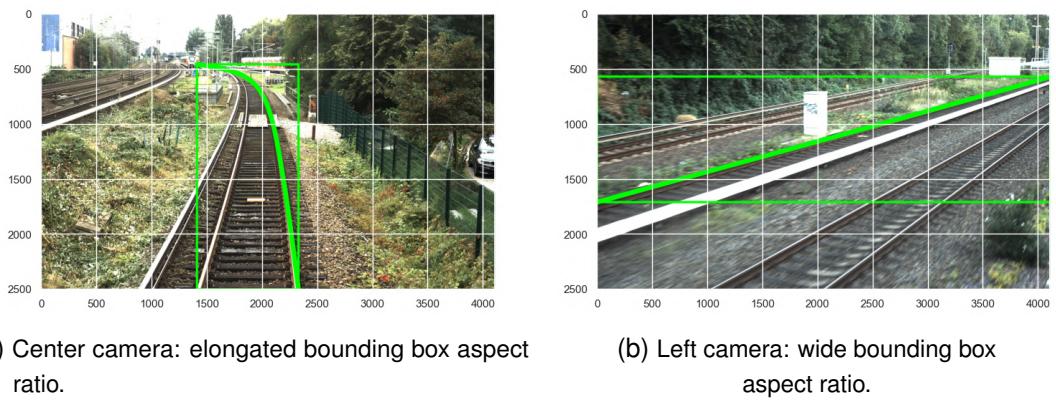


Figure 19: Different bounding box aspect ratios depending on camera orientation.

Figure 20 gives the distribution of the bounding box aspect ratios for the different camera orientations in the OSDaR23 dataset. The histogram confirms the idea that different sensors have different aspect ratios for the bounding boxes. For more details see Appendix A.3.

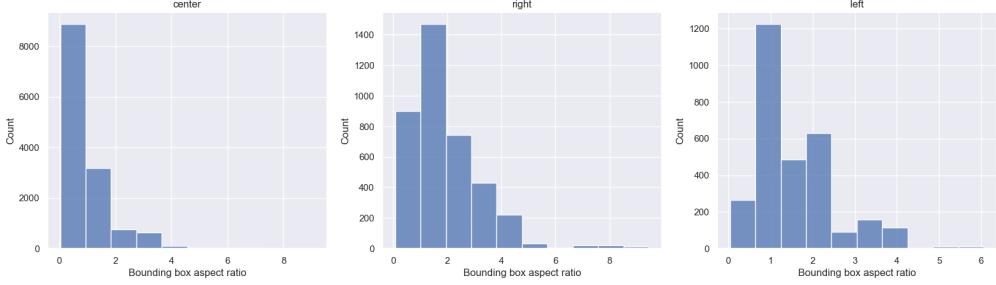


Figure 20: Histogram of bounding box aspect ratios by camera orientation.

The aspect ratios are used to remove all lines detected by FLD that have a bounding box aspect ratio smaller than the x^{th} quantile or larger than the $1 - x^{th}$ quantile.

In a final step, we apply Connected Component Labeling (CCL) to remove noise or segments that are false positives (OpenCV, 2024a). The goals of CCL are to identify and label connected regions in a binary image where pixels are either tracks or background. After connected components are labeled in the image, we remove all components that are $x\%$ smaller than the largest component.

The entire process is illustrated in Figure 21. The final result showcases that the approach is a heuristic that might filter out true positive labels.

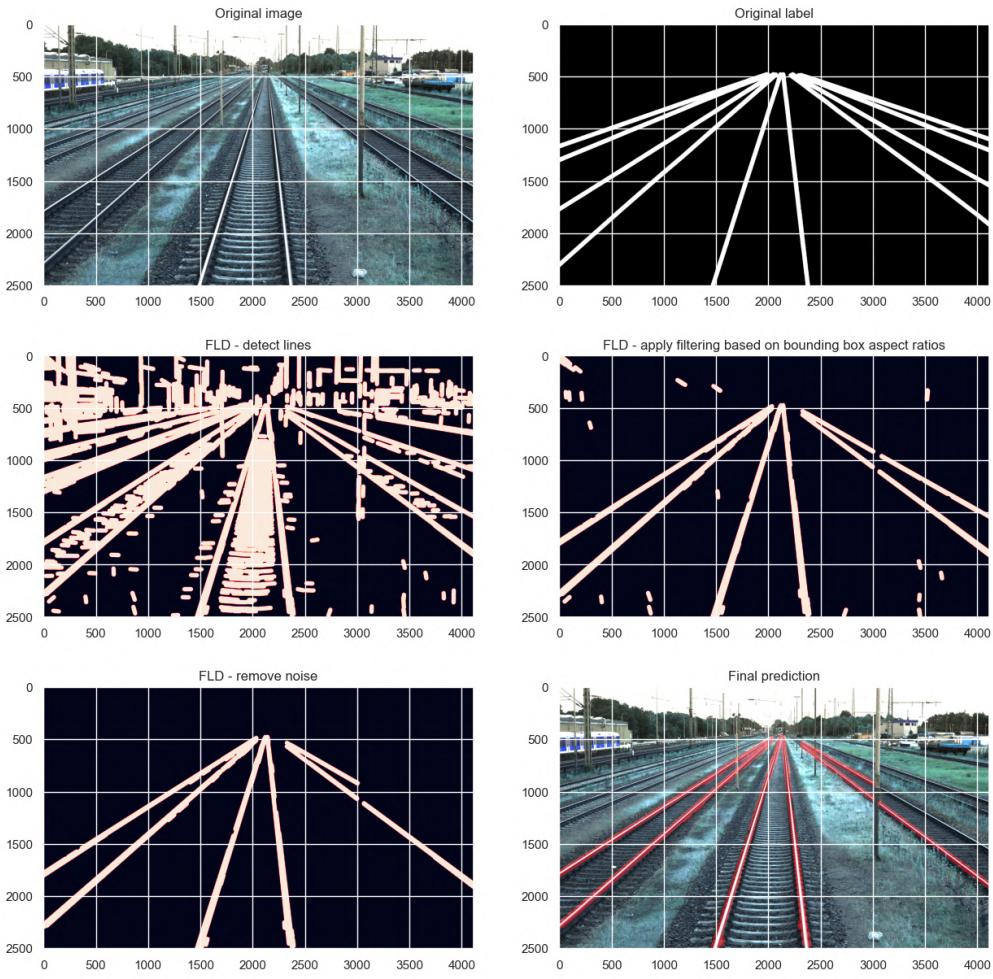


Figure 21: Baseline approach based on fast line detection and post-processing.

4.4.3 Parameter tuning

The performance of our traditional rail detection algorithm depends on a number of parameters. In order to choose the best parameter setting for each combination of sensor and orientation, we tune the parameters according to the grid search technique. Grid search involves specifying a grid of parameter values and exhaustively evaluating the performance of the algorithm for each combination of values. The parameter combination that yields the best performance is selected. Grid search is a simple technique that thoroughly explores the parameter space but it is also computationally expensive in multi-dimensional spaces.

In total, we test 22.680 parameter settings on 196 images¹ from the OSDaR23 dataset. The images are sampled randomly from the train set such that there is one image per sensor per video sequence. The configuration of the grid search is given in Table 2.

Parameter	Description	Values
Sensors	Combination of sensor and orientation	rgb_highres_left, rgb_highres_right, rgb_highres_center, rgb_left, rgb_right, rgb_center, ir_left, ir_right, ir_center ir_left, ir_right, ir_center
Kernel	Kernel for Gaussian blur	0, 3, 5, 7
Length Threshold	Segment shorter than this will be discarded	25, 50, 100, 200, 300
Canny Threshold 1	First threshold for hysteresis procedure in Canny; second threshold is threshold 1 times 3	5, 10, 20, 50, 100, 150
Aspect Ratio Quantile	Filtering out lines depending on bounding box aspect ratio quantile	1, 5, 10
Noise Threshold	Removing small segments that are classified as noise	0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3

Table 2: Sensor parameters.

Table 3 presents the best parameter configurations based on the average Dice score. The varying parameter settings across different sensors demonstrate the algorithm's sensitivity to the selected parameters. The left-oriented camera images seem to benefit from a higher level of blurring with a kernel size of seven. The filtering criteria for lines with uncharacteristic bounding box aspect ratios appear to be relatively loose in most cases – only the most extreme outliers need to be removed. Left and right-oriented high-resolution camera images, on the other hand, seem to require a more restrictive filtering. The "Avg. Time" column indicates the average inference time in seconds. The inference times seem to be strongly correlated with the entropy metrics discussed in Section 3.1.3; higher entropy values tend to be associated with longer inference times.

Sensor	Kernel	Length Threshold	Aspect Ratio Quantile	Canny Th1	Canny Th2	Noise Threshold	Avg. Time (sec)
rgb_highres_center	5	50	1	50	150	0.25	0.052
rgb_highres_right	0	100	5	10	30	0.25	0.074
rgb_center	3	100	1	10	30	0.2	0.014
rgb_highres_left	7	50	5	50	150	0.25	0.069
rgb_right	3	50	1	5	15	0.1	0.032
ir_left	7	25	1	5	15	0.2	0.003
rgb_left	7	100	1	20	60	0.25	0.026
ir_center	3	25	1	5	15	0.2	0.003
ir_right	5	25	1	10	30	0.2	0.004

Table 3: Best parameter settings and associated performance metrics.

¹The idea is to select one image for each combination of sensor, sensor orientation, and video sequence in the train set (i.e., $3 \times 3 \times 33 = 297$). However, there are combinations with no track labels.

4.5 Deep learning-based segmentation

In this section, we will describe the deep learning-based segmentation approach in more detail. The applied model is YOLOv8 which is the latest version of YOLO models by Jocher et al. (2023). YOLOv8 is licensed under the GNU General Public License, which authorizes the users to freely share, modify and distribute the software. The accessibility of the code as well as its high performance on many applications led to the formation of a large community and the availability of numerous resources for developing and fine-tuning YOLO models.

YOLOv8 is used to detect objects in images with a single forward pass through the network. The models are pre-trained on large datasets such as COCO and ImageNet which enables them to predict everyday objects (Krishnakumar, 2023). By following the transfer learning technique, the model can be fine-tuned on new datasets related to the problem at hand (i.e., rail track detection).

The model architecture is presented in Appendix A.4. YOLOv8-Seg model is an extension of the YOLOv8 object detection model that performs semantic segmentation of the input image. The model has been shown to achieve state-of-the-art results on a variety of object detection and semantic segmentation benchmarks while maintaining high speed and efficiency (Krishnakumar, 2023).

YOLOv8-seg uses a weighted loss function comprising bounding box loss, objectness loss, and segmentation loss (Jocher, 2023). Bounding box loss calculates the error between the geometry of the predicted and the ground truth box. It is a measure of how well the model predicts the size and location of the bounding boxes. Objectness loss determines how confident the model is about the presence of an object in the bounding box. It compares the probability of an object being present in the prediction with the ground truth value. Finally, segmentation loss quantifies how close the predicted segmentation map is to the ground truth map. It measures how effectively the model performs the semantic segmentation task.

4.5.1 Models

In order to identify the best model setup for the rail track detection task, we experiment with 18 different models that are validated against different datasets. The models differ in the data the model is trained on, the size of the input image, and the labeling approach. All models are trained for at most 300 epochs. However, the optimization process stops

if there has been no improvement in the last 50 epochs. Hyper-parameters are either set to default values or automatically chosen in the initialization phase by YOLO.

Table 4 summarizes the configuration of the different models. The “Epochs” column gives the actual number of performed epochs and “Duration” is the wall-clock time for the training process with unified labels and separate labels, respectively. The remaining specifications in the table are the same regardless of the labeling approach. The column “Train Set” gives the set of images that are used to train the model. “Image Size” is either set to 640 or 1280 in order to assess the influence of the image size in the prediction results; the model resizes the input image such that the larger dimension is scaled to the given number of pixels while maintaining the original aspect ratio (e.g., if image size is set to 640 then the RGB low-resolution images are resized from 2464x1600 to 640x416 as the image length and width have to be a multiples of 32). The batch size is selected automatically by YOLO such that the model can best utilize the available GPU memory without running into “out of memory” situations. The “Optimizer” column gives the optimization algorithms used for training the neural networks. The optimizer is selected automatically. SGD (Stochastic Gradient Descent) is a basic optimization algorithm used to minimize the loss function during the training of neural networks. In each iteration, SGD randomly selects a subset of training examples (also referred to as mini-batch) to compute the gradient estimate which supports faster convergence (Bottou, 2010). AdamW (Adam with Weight Decay) is a variant of the Adam optimizer that includes an additional term for weight decay regularization. AdamW combines the advantages of both Adam and weight decay regularization which helps prevent overfitting by penalizing large parameter values (Kingma and Ba, 2014).

Train Set	Image Size	Batch Size	Optimizer	Unified labels		Separate labels	
				Epochs	Duration (HH:MM:SS)	Epoch	Duration (HH:MM:SS)
OSDaR23 RGB	640	23	AdamW	121	00:37:57	93	00:35:29
OSDaR23 infrared	640	23	AdamW	196	00:51:51	223	01:01:10
OSDaR23 RGB	1280	5	AdamW	117	01:39:58	247	03:54:41
OSDaR23 RGB + Railsem19	640	23	SGD	138	02:44:37	300	08:45:13
OSDaR23 high-res	1280	5	AdamW	81	04:20:46	119	17:36:20
OSDaR23 high-res + Railsem19	640	23	SGD	110	13:36:26	300	60:03:41
OSDaR23 high-res	640	23	AdamW	130	13:53:44	285	43:27:41
All OSDaR23	640	23	SGD	189	23:27:11	172	36:45:33
All OSDaR23 + Railsem19	640	23	SGD	224	23:42:53	300	64:51:24

Table 4: Training details.

When looking at the table, it becomes apparent that the size of the input images has a large impact on the batch size. With an image size of 640, 23 images can be

processed per iteration. If the image size is set to 1280 pixels, only 5 images can be processed at once with the given computational resources. The automatic selection of optimizers seems to depend on whether or not the Railsem19 images are added to the train set. With Railsem19 data, YOLO selects the SGD optimizer, without Railsem19 data the AdamW optimizer is used. In terms of computational time, it appears that the high-resolution images significantly influence the duration of processing. Among the experiments with an image size of 640, the presence of high-resolution images in the training set causes at least a five-fold increase in the computation time per epoch. The labeling approach is another factor that has a large effect on the computational time; training the model with all images (OSDaR23 and RailSem19) with separate labels took almost three days, whereas the training with unified labels stopped after less than a day. Also, all models with unified labels stopped early as no improvement was observed for 50 epochs. With separate labels the maximum number of iterations was reached in models that incorporated the RailSem19 dataset.

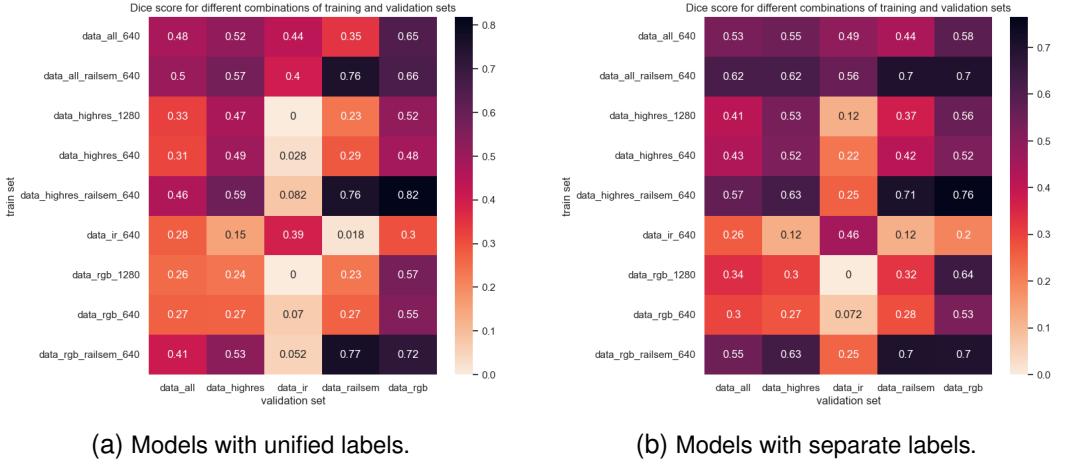
Figure 22 presents the results of the experiments with unified labels on the left and with separate labels on the right. On the y-axis, we list the nine models introduced above. The x-axis shows the validation sets on which the models are tested. The models are compared based on the average Dice score.

The model that performs best on all OSDaR23 images is data_all_railsem_640, on OSDaR23 high-resolution and low-resolution images data_highres_railsem_640 and on OSDaR23 infrared images data_all_640 with unified labels and data_all_railsem_640 with separate labels. This means that all models benefit from adding images from the Railsem19 dataset to the training set, except when applied to the infrared images with unified labels. This result comes at no surprise as the Railsem19 dataset comprises a large variety of different images and deep learning models are known to perform better with more training data. The lack of generalization of the models trained exclusively on the OSDaR23 data becomes apparent when validated against the Railsem19 dataset: the best dice score is 0.35 with the data_all_640 model compared to 0.76 with the data_highres_railsem_640 model when labels are unified; with separate labels the difference is 0.44 versus 0.71.

It is also interesting to observe the models that are validated against infrared images. While all models with unified labels fail to predict tracks in infrared images except of models that are trained on infrared images, models trained exclusively on infrared images can predict tracks in RGB images with a Dice score of 0.3. This indicates that training on infrared images supports the prediction on RGB images but not the other way round. The situation is different when each track is labeled separately. Even if the training set

does not contain infrared images, tracks can be predicted in infrared images with a Dice score of 0.25.

In general, the results clearly indicate that the performance of the segmentation approach depends on the availability of appropriate training data. The comparison between the different labeling approaches is not straightforward because the Dice score is calculated differently. We will perform a comparison based on the same metrics in Section 5.



(a) Models with unified labels.

(b) Models with separate labels.

Figure 22: Models with different train and validation sets.

The comparison between using 1280 pixel images as input and 640 pixel images reveals that using larger images has no clear advantage on the solution quality. In certain cases (e.g., data_railsem) the Dice score even decreases when using 1280 pixel images rather than 640 pixel images which might be caused by the smaller batch size that fits into the memory in each training step. What is more, using larger images considerably increases the inference speed, which might become a bottle-neck in real-life applications. Table 5 shows that using large images as input can increase the average inference speed by a factor of around 4.

Train Set	Inference speed unified labels [ms]		Inference speed separate labels [ms]	
	data_highres	data_rgb	data_highres	data_rgb
data_highres_1280	7.286	7.478	10.768	11.624
data_highres_640	1.696	1.745	4.138	3.001
data_rgb_1280	5.371	6.755	8.306	8.869
data_rgb_640	2.019	1.867	1.716	1.931

Table 5: Inference speed in milliseconds.

Going forward, we will focus on the models `data_highres_railsem_640` and `data_all_640` with unified labels, and models `data_highres_railsem_640` and `data_all_railsem_640` with separate labels for further analysis and refinement as these models provided the best results on the relevant OSDaR23 validation sets. The detailed training results are given in Appendix A.4.

4.5.2 Hyperparameter tuning

Hyperparameter tuning refers to the process of selecting the optimal hyperparameters for a neural network model. Hyperparameters cannot be directly learned from the training data but are high-level settings that are provided as an external input to the model. The goal of parameter tuning is to find the combination of hyperparameters that results in the best performance. Ultralytics YOLO enables hyperparameter tuning by genetic algorithms.

Parameter tuning in YOLO is an iterative process where in each iteration the fitness of the current setting is assessed by training the model for a given number of epochs. We set the number of iterations to 300 and the number of epochs per iteration to 30. Preliminary experiments revealed that the tuning process is computationally expensive. Therefore, the tuning is performed on 10% of the training data. Table 6 summarizes the tuning runs with unified labels. Despite the reduced training set, the tuning on the high-resolution OSDaR23 images plus Railsem19 images took 146 hours with the SGD optimizer and was aborted after 148 hours with the AdamW optimizer.

Model	Optimizer	Duration [h]
<code>data_highres_railsem_640</code>	AdamW	148 ^a
	SGD	146
<code>data_all_640</code>	AdamW	50
	SGD	51

^aTuning run was aborted by the system after 160 iterations.

Table 6: Parameter tuning with unified labels.

Table 7 compares the results with unified labels that are generated by automatically setting the parameters by YOLO, tuning with SGD, and tuning with AdamW. The `data_all_640` model is used as a basis for the infrared images and `data_highres_railsem_640` model is used for the RGB images. The results indicate that the automatically chosen parameters outperform the tuned parameters on two out of four tests. This is a surprising result, but it can probably be explained by the reduced training set that was used during the parameter tuning. For example, if the subset of the

training data used for tuning contained less infrared images relative to the entire training set, then the parameters of the data_all_640 model would be tuned towards performing better on RGB images. Detailed results of the tuning are given in Appendix A.4.5.

Given the good overall performance of the automatically tuned parameters and the poor performance of the tuned parameters on the data_rgb set, we use the models with the automatically chosen parameters for the final analysis. In the same sense, the parameter tuning was omitted for the models with separate labels.

Model	Validation Set	Dice			SGD vs.	AdamW vs.
		Auto	Tuned SGD	Tuned AdamW	Auto [%]	Auto [%]
data_all_640	data_ir	0.440	0.425	0.375	-3.324	-14.704
data_highres_railsem_640	data_highres	0.587	0.596	0.558	1.480	-4.913
	data_railsem	0.758	0.765	0.766	0.879	1.057
	data_rgb	0.818	0.774	0.752	-5.440	-8.140

Table 7: Comparison of automatically chosen parameters and tuned parameters based on Dice score with unified labels.

5 Results

In this section, we analyze the results of the introduced solution approaches and investigate options for post-processing and integration into existing tools for rail-track detection.

5.1 Performance analysis

In this section, we perform a detailed analysis of the proposed solution approaches on the OSDaR23 test set. The test set is not used during the model development and training process, so the results provide an unbiased performance evaluation on unseen data. All approaches, FLD, YOLOv8 with unified labels, and YOLOv8 with separate labels are used to predict tracks in the test images. The evaluation is based on the Dice score of the resulting segmentation mask.

YOLOv8 uses different methods to evaluate the performance during the validation (i.e., on validation set) and prediction (i.e., on test set). The validation method generates

detections on padded images, while the predict method does not. This means that the two methods are using slightly different versions of the same image when generating predictions. In order to enable a fair comparison between YOLOv8 and FLD, we evaluate the performance of both approaches by using the same function.

First, we compare the YOLOv8 models with unified labels and with separate labels. Figure 23 visualizes the distribution of the Dice scores by sensor type and orientation in a violin plot – a visualization that combines aspects of a box plot and a kernel density plot¹. Both approaches seem to provide results of similar quality. For the left oriented RGB camera and the right oriented high-resolution camera, there is a clear advantage of the separate labeling compared to the unified labeling. For the left oriented infrared camera, the unified labels provide better results.

As both models perform similarly well, we continue the analysis with the model using separate labels, as it offers advantages when applying the results in real-world scenarios. Figure 24 illustrates the result of the two labeling approaches. On the top, we give the segmentation result when the labels are unified. In this case, the output is a single bounding box and a single mask marking all pixels that are classified as track. On the other hand, the example on the bottom shows the result when the model is trained with separate labels. In this case, the model provides a separate bounding box and mask for each track in the image which makes it easier to interpret the environment.

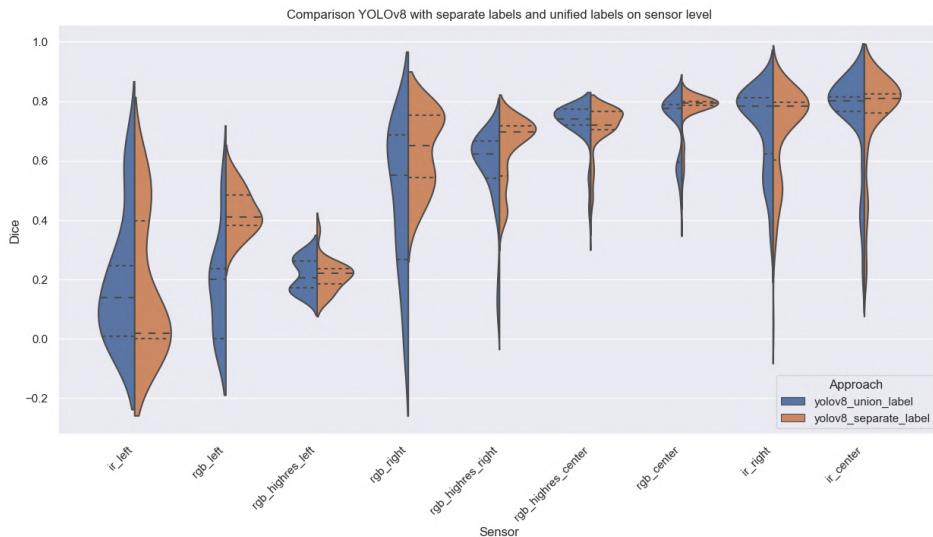


Figure 23: Comparison between YOLOv8 models with unified labels and separate labels by sensor type.

¹The negative values in the violin plot are caused by the kernel density estimation.



(a) Prediction image with unified labels.



(b) Prediction mask with unified labels.



(c) Prediction image with separate labels.



(d) Prediction mask with separate labels.

Figure 24: Example results of different labeling approaches.

Figure 25 shows the comparison between FLD and YOLOv8 (with separate labels) based on the Dice score where each data point represents one image. The sensor types are distinguished by color. The dashed red line represents the indifference line where scores along this line are the same for YOLOv8 and FLD. If a point is above the line then YOLOv8 performed better on the given image; if a point is below the line, then FLD outperformed YOLOv8. It is easy to see that YOLOv8 performs better than FLD on most images. However, in certain instances FLD achieves better results, e.g., on infrared images. A closer look reveals that most clusters of images where YOLOv8 provides worse predictions than FLD are frames from the same sequence.

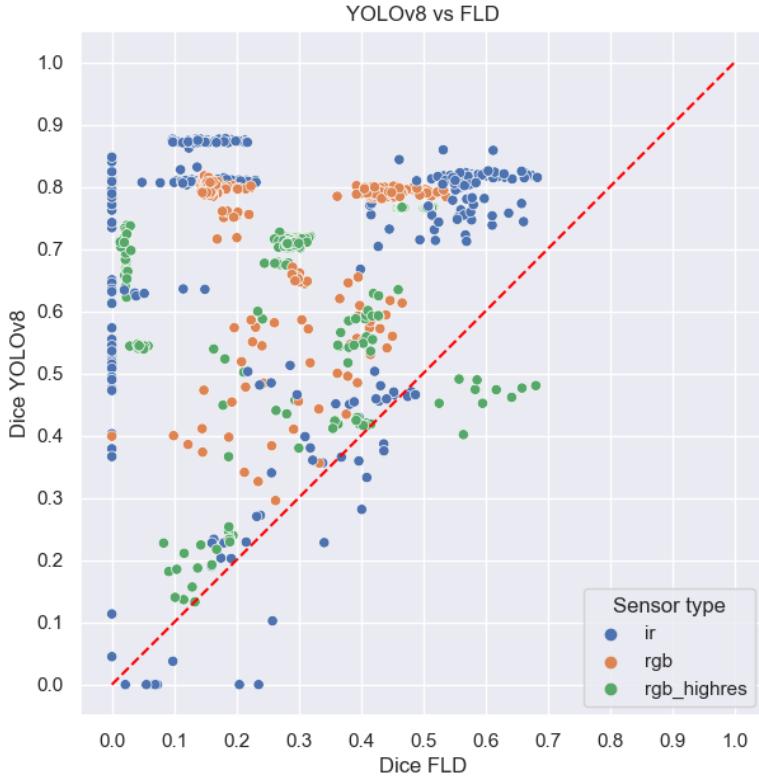


Figure 25: Comparison between YOLOv8 (separate labels) and FLD based on Dice score. Sensor types are distinguished by color.

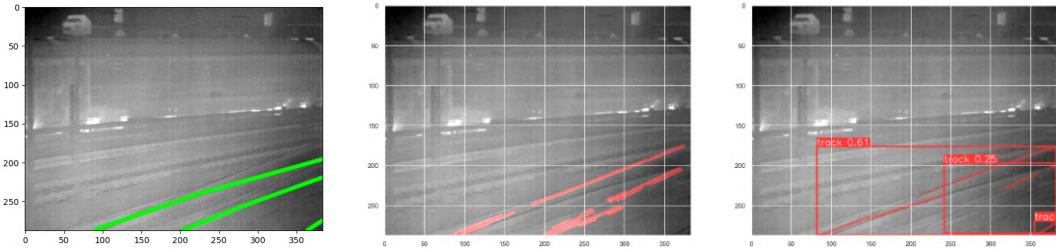
An example is given in Figure 26. The ground truth image is given on the left; the labels are colored green². Predictions by FLD and YOLOv8 are colored red.

Similarly, the points in the cluster of high-resolution images, on which FLD performs better, are all frames from the same sequence. The prediction of YOLOv8 and of FLD on one of these frames is given in Figure 27.

Another interesting observation is that infrared images have the highest Dice scores among all images segmented by YOLOv8. Figure 28 illustrates the difference between FLD and YOLOv8 on an infrared image.

In Section 4.4.3, we have seen that the image entropy has an effect on the computation time. In Figure 29, we examine the relation between Dice and entropy score; the solution approaches are distinguished by color. There is no clear trend in the scatter plot, i.e., there is no linear relationship between Dice score and entropy. However, both approaches seem to struggle with low entropy images that have uniform intensity

²Note that the provided labeling is not optimal as certain track segments are not marked as tracks.



(a) Image with ground truth label in green.
(b) FLD provides better prediction for main tracks; the side tracks are only partially segmented.
(c) YOLOv8 has many False Negative pixels and low confidence in the result (confidence values are given for each bounding box).

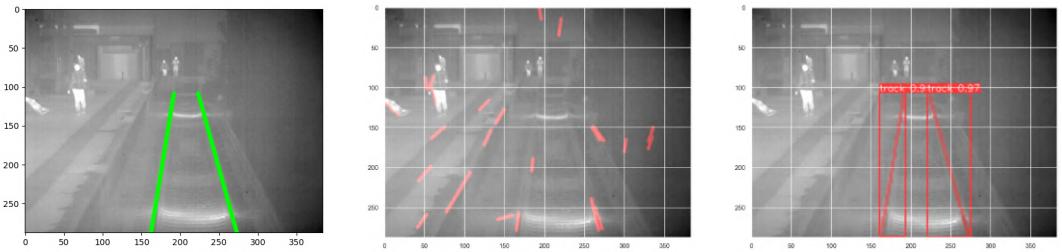
Figure 26: Example image where FLD performs better than YOLOv8; predictions are colored red.



(a) Image with ground truth label in green.
(b) FLD provides good prediction for main tracks; the side tracks are only partially segmented.
(c) YOLOv8 has many False Negative pixels – confidence values are given for each bounding box.

Figure 27: Example where FLD slightly outperforms YOLOv8.

(which might be the case with infrared images – see Figure 8a). Figure 30 shows the relation between image brightness and the Dice score achieved by FLD and YOLOv8, respectively. Image brightness does not seem to have a significant effect on the result. Even in very dark images YOLOv8 is able to generate predictions with Dice scores of around 0.8.



(a) Image with ground truth label in green.
(b) FLD approach cannot detect tracks efficiently in the example image.
(c) YOLOv8 provides precise segmentation with high confidence.

Figure 28: Extreme example where YOLOv8 clearly outperforms FLD.

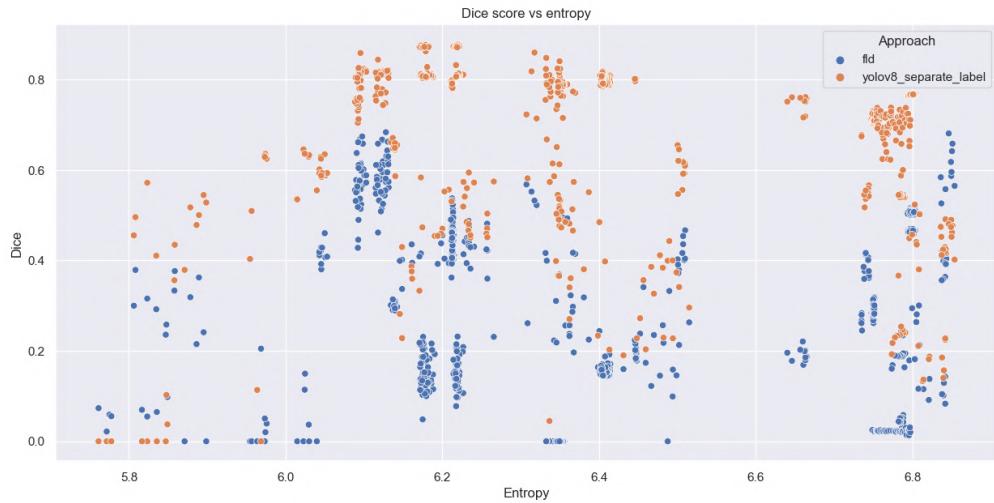


Figure 29: Dice scores vs. entropy scores – FLD and YOLOv8 are distinguished by color.

Figure 31, evaluates and compares FLD and YOLOv8 on a sensor level. The Dice scores are visualized in a violin plot. The diagram supports the conclusion from Figure 25: YOLOv8 performs better than FLD in most of the cases. YOLOv8 performs particularly well on infrared images. Good results are achieved with forward-facing sensors. In particular, FLD performs best on forward-facing RGB cameras, but performs poorly when the cameras are left or right oriented. YOLOv8 is more robust to the sensor type and orientation. Segmenting tracks on images of left oriented cameras seems to be particularly challenging which might be an outcome of the right-hand traffic principle, which is mostly followed in Germany.

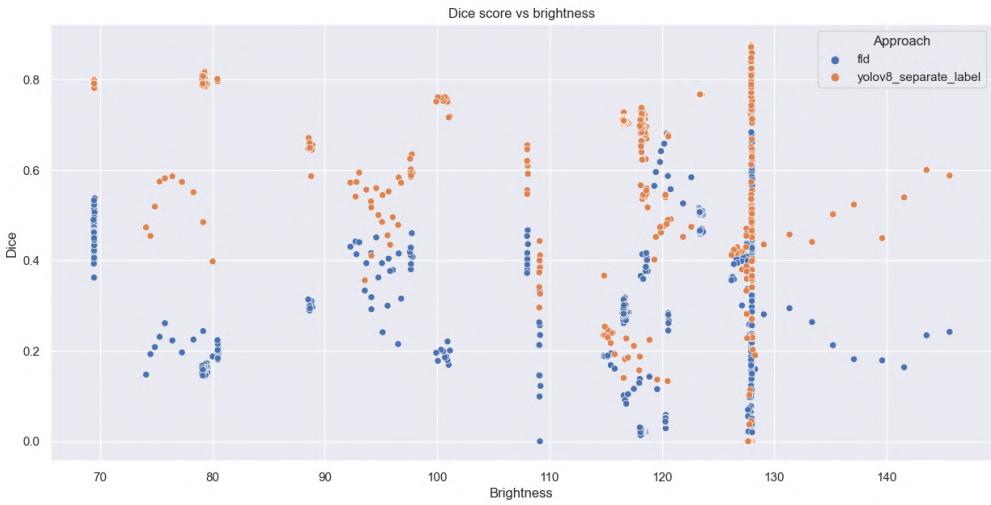


Figure 30: Dice scores vs. brightness scores – FLD and YOLOv8 are distinguished by color.

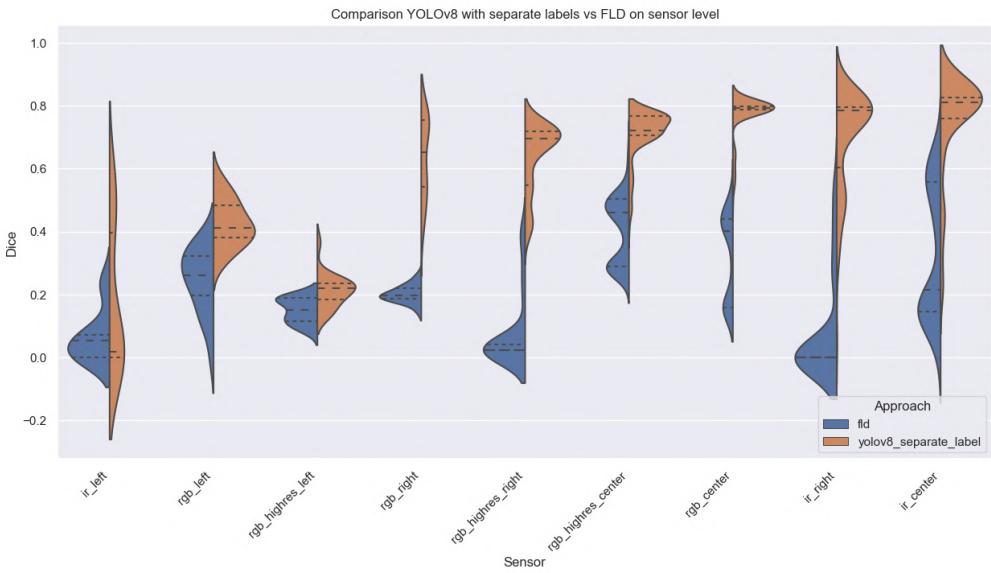


Figure 31: Violin plot indicating the difference between YOLOv8 and FLD on sensor level.

5.2 Post-processing

Representing the identified tracks as a polylines as opposed to segmentation masks can have several practical advantages in the area of rail track detection. For example, polylines given as an array of points offer a concise depiction of the detected rail tracks

which facilitates the interpretation of the result. Additionally, using polylines reduces data volumes compared to segmentation masks which is advantageous if the bandwidth of the computational resources is limited.

Training the model with separate labels for each track allows us to identify tracks individually in a post-processing step. Each mask is converted into one polyline by applying a traditional line detector. Figure 32 illustrates the idea. The image on the left shows the original input with the overlaid mask given in red. The image in the middle shows the segmentation mask which is generated by combining the four individual masks – one for each track. The image on the right displays the result where each track is marked by a polyline with a different color.

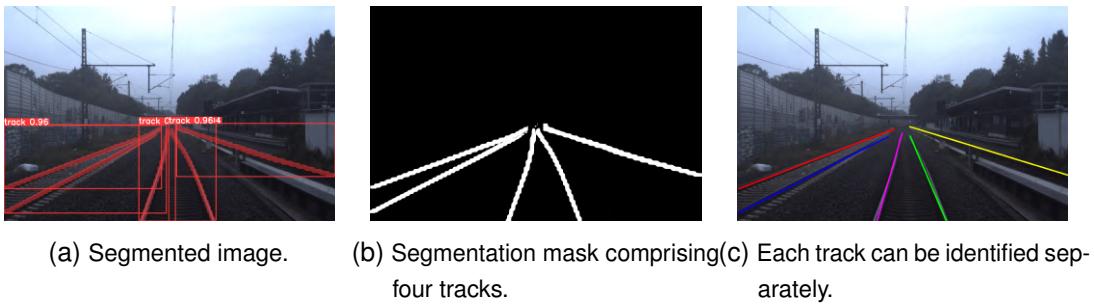


Figure 32: Converting masks to polylines.

5.3 Integration into real-world applications

The ultimate goal of the proposed solution approaches is to integrate the image segmentation model into a live video stream.

Once the image segmentation model has been trained, the camera output is fed into the model. Each frame of the video stream is preprocessed to meet the input requirements of the segmentation model (e.g., the image resized). The frames are then analyzed by the segmentation model to obtain the segmentation masks and labeled regions. The segmentation masks can be overlaid onto the original video frames to visualize the segmentation results and the processed frames can be displayed in real-time. Alternatively, the masks can be used as an input into existing rails track detection approaches that incorporate domain knowledge.

As a proof of concept, we integrate the YOLOv8 model with a rail track analysis algorithm proposed by Capriotti (2024). The pseudocode of the algorithm is given in Algorithm 1. The first step is to capture the frames of the video stream (line 2). Each

frame is preprocessed and analyzed by the YOLO model, resulting in a segmentation mask and an image with the segmentation overlay (line 3). The mask and the details about the camera setup are used to applying the homography technique to transform the frames into a bird's-eye view (line 4). Homography is the process of transforming images taken from the camera's perspective to a common coordinate system. In this system, the scene is viewed from above, as if the camera were positioned directly overhead. In order to generate the bird-eye view, it is essential to calibrate the camera by estimating parameters such as focal length, principal point, and lens distortion coefficients. This information is used to accurately transform the points between the camera images and the new coordinate system. The bird-eye view of the mask is then used to extract potential lines that might refer to rail tracks (line 5). There are different approaches to extract lines from the mask. The approach implemented in our experiment is based on finding the contours of the mask, splitting the image into different horizontal segments, calculating the mean point among all points within the contour per segment, and fitting a line through those points. In line 6, we incorporate explicitly available knowledge into the algorithm to analyze the bird-eye view image and identify the main track and the secondary tracks. For example, the tracks are always parallel from a bird-eye view and the standard gauge (i.e., the distance between the left and the right track) is always 1435mm. This information is used to exclude false positive segmentation results and pair tracks that belong together.

Algorithm 1 Rail track detection

```

1: procedure IDENTIFY_TRACKS(video_stream, yolo_model, camera_setup)
2:   while f  $\leftarrow$  capture_frame(video_stream) do
3:     mask, image_with_overlay  $\leftarrow$  yolo_model(f)
4:     birdseye_view  $\leftarrow$  apply_homography(mask, camera_setup)
5:     track_lines  $\leftarrow$  extract_lines(birdseye_view)
6:     main_tracks, secondary_tracks  $\leftarrow$  process_track_lines(track_lines)
7:   return main_tracks, secondary_tracks

```

The process is illustrated in Figure 33. The image on the left shows the result provided by YOLO. The figure in the middle shows the output of Algorithm 1: the perspective projection of the camera is illustrated in magenta; the main tracks are shown in red and the secondary tracks in blue, respectively. The image on the right shows the process of generating the bird-eye view, finding the average contour points (marked by green circles), and fitting potential track lines.

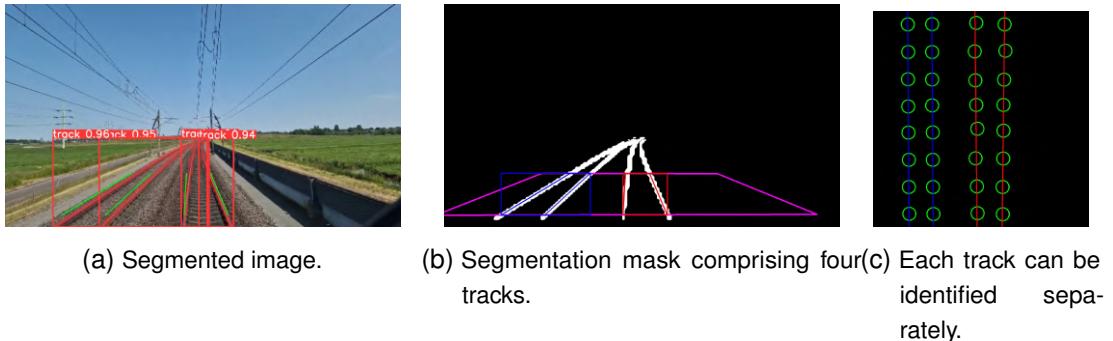


Figure 33: Illustration of “rail track detection” algorithm.

Several optimization strategies can be implemented to improve the real-time performance of video processing and enable a seamless segmentation with computationally intensive operations. For example, parallel processing techniques such as multiprocessing can be utilized to concurrently handle different tasks simultaneously (e.g., segmentation processing, frame capturing, and image preprocessing). Hardware acceleration, such as GPU utilization, can significantly speed up inference by leveraging parallel processing capabilities. Additionally, selectively skipping frames or processing frames at a lower frequency can reduce the computational load, especially in scenarios with relatively static scenes that often occur on straight tracks. Furthermore, the segmentation model itself can be optimized for faster inference.

6 Conclusion

This master’s thesis addresses a challenge in the area of railway automation. The focus is on multi-sensor rail track detection within the context of Automatic Train Operations (ATO). By leveraging deep learning techniques, particularly semantic segmentation, the thesis aims to increase the accuracy and robustness of track detection systems.

The research compares deep learning-based segmentation with traditional non-AI-based methods and evaluates the effectiveness of both approaches on different sensor inputs and orientations. By conducting a comprehensive performance evaluation and testing the integrating of the developed model into real-world applications, valuable insights are gained into the practical implications of automated track detection systems.

The results demonstrate the advantages of deep learning-based approaches, particularly YOLOv8, over traditional methods in most scenarios. YOLOv8 performs robustly across various sensor types and orientations, with infrared images yielding the highest Dice scores. Furthermore, the results highlight the challenges associated with low entropy images and the impact of sensor orientation on segmentation accuracy. Deep learning approaches have the disadvantage that it is difficult to incorporate expert knowledge. By borrowing from traditional track-detection approaches and post-processing the segmentation results, we successfully mitigate false positive classifications and achieve higher accuracy in identifying tracks.

Overall, this research contributes to the field of railway automation by providing valuable insights, methodologies, and tools for improving the safety and efficiency of automatic train operations.

Future research could involve applying other deep learning techniques for solving the rail detection problem instead of semantic segmentation. Considering that the desired output in many cases is polylines marking tracks, it would be interesting to adapt the YOLinO approach proposed by Meyer et al. (2021) for lane detection. The approach proposed in this thesis performs well on images from infrared sensors. Therefore, research could focus on more detailed analysis on track detection in adverse weather and low lighting conditions. Another interesting stream of research is the combination of CNNs with recurrent neural network (RNN) architecture designed to model sequential data. Capturing temporal dependencies between video frames could overcome the limitation of analyzing each frame independently, regardless of the sequence in which the frames are generated. Finally, for real-world applications, further research could focus on combining traditional segmentation approaches and deep learning-based approaches to increase the overall performance of the system.

Bibliography

- Emilio J. Almazàn, Ron Tal, Yiming Qian, and James H. Elder. Mcmlsd: A dynamic programming approach to line segment detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5854–5862, 2017. doi: 10.1109/CVPR.2017.620.
- ALSTOM Transport SA. Autonomous mobility: The future of rail is automated, 2021. URL <https://www.alstom.com/autonomous-mobility-future-rail-automated>.
- Mostafa Arastounia. Automated recognition of railroad infrastructure in rural areas from lidar data. *Remote Sensing*, 7(11):14916–14938, 2015. ISSN 2072-4292. doi: 10.3390/rs71114916. URL <https://www.mdpi.com/2072-4292/7/11/14916>.
- Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.
- Daniele Capriotti. Track analysis, 2024.
- Jieren Cheng, Hua Li, Dengbo Li, Shuai Hua, and Victor S. Sheng. A survey on image semantic segmentation using deep learning techniques. *Computers, Materials & Continua*, 74(1):1941–1957, 2023. ISSN 1546-2226. doi: 10.32604/cmc.2023.032757. URL <http://www.techscience.com/cmc/v74n1/49879>.
- M Clark, D.M McCann, and M.C Forde. Infrared thermographic investigation of railway track ballast. *NDT & E International*, 35(2):83–94, 2002. ISSN 0963-8695. doi: [https://doi.org/10.1016/S0963-8695\(01\)00032-9](https://doi.org/10.1016/S0963-8695(01)00032-9). URL <https://www.sciencedirect.com/science/article/pii/S0963869501000329>.
- Deutsche Bahn AG. Automatic train operation (ato), 2022. URL <https://digitale-schiene-deutschland.de/en/Automatic-Train-Operation>.

Deutsche Bahn AG. First freely available multi-sensor data set for machine learning for the development of fully automated driving: Osdar23, 2023. URL <https://digitale-schiene-deutschland.de/en/news/2023/OSDaR23-multi-sensor-data-set-for-machine-learning>.

Tausif Diwan, G. Anirudh, and Jitendra V. Tembhurne. Object detection using yolo: challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, 82(6):9243–9275, 03 2023. ISSN 1573-7721. doi: 10.1007/s11042-022-13644-y. URL <https://doi.org/10.1007/s11042-022-13644-y>.

Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972. ISSN 0001-0782. doi: 10.1145/361237.361242. URL <https://doi.org/10.1145/361237.361242>.

Europe's Rail Joint Undertaking. Innovation in the spotlight: Towards unattended mainline train operations (ato goa 4), 2019. URL <https://rail-research.europa.eu/highlight/innovation-in-the-spotlight-towards-unattended-mainline-train-operations-ato-goa4/>.

Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang, and Tao Yuan. Benchmark analysis of yolo performance on edge intelligence devices. *Cryptography*, 6(2), 2022. ISSN 2410-387X. doi: 10.3390/cryptography6020016. URL <https://www.mdpi.com/2410-387X/6/2/16>.

Xavier Giben, Vishal M. Patel, and Rama Chellappa. Material classification and semantic segmentation of railway track images with deep convolutional neural networks. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 621–625, 2015. doi: 10.1109/ICIP.2015.7350873.

Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, 2010. doi: 10.1109/TPAMI.2008.300.

Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. LSD: a Line Segment Detector. *Image Processing On Line*, 2:35–55, 2012. <https://doi.org/10.5201/ipol.2012.gjmr-lsd>.

Jeanine Harb, Nicolas Rébena, Raphaël Chosidow, Grégoire Roblin, Roman Potarusov, and Hatem Hajri. Frsign: A large-scale traffic light dataset for autonomous trains. *arXiv preprint arXiv:2002.05665*, 2020.

International Energy Agency. The future of rail, 2019. URL <https://www.iea.org/reports/the-future-of-rail>.

Dewan Md Zahurul Islam, Stefano Ricci, and Bo-Lennart Nelldal. How to make modal shift from road to rail possible in the european transport market, as aspired to in the eu transport white paper 2011. *European transport research review*, 8(3):1–14, 2016.

Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2022.01.135>. URL <https://www.sciencedirect.com/science/article/pii/S1877050922001363>. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19.

Glenn Jocher. What is the loss used for YOLOv8-seg?, 2023. URL <https://github.com/ultralytics/ultralytics/issues/3882>.

Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO. <https://github.com/ultralytics/ultralytics>, January 2023. AGPL-3.0 License.

Fatih Kaleli and Yusuf Sinan Akgul. Vision-based railroad track extraction using dynamic programming. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, 2009. doi: 10.1109/ITSC.2009.5309526.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Mukilan Krishnakumar. A Gentle Introduction to YOLOv8, 2023. URL <https://wandb.ai/mukilan/wildlife-yolov8/reports/A-Gentle-Introduction-to-YOLOv8--Vmldzo0MDU5NDA2>.

B. Le Saux, A. Beaupère, A. Boulch, J. Brossard, A. Manier, and G. Villemin. Railway detection: From filtering to segmentation networks. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 4819–4822, 2018. doi: 10.1109/IGARSS.2018.8517865.

Jin Han Lee, Sehyung Lee, Jongwoo Lim, and Il Hong Suh. Outdoor place recognition in urban environments using straight lines. 06 2014. doi: 10.1109/ICRA.2014.6907675.

Xinpeng Li and Xiaojiang Peng. Rail detection: An efficient row-based network and a new benchmark. In *Proceedings of the 30th ACM International Conference on*

Multimedia, MM '22, page 6455–6463, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392037. doi: 10.1145/3503161.3548050. URL <https://doi.org/10.1145/3503161.3548050>.

Chao Ma and Mei Xie. A method for lane detection based on color clustering. In *2010 Third International Conference on Knowledge Discovery and Data Mining*, pages 200–203, 2010. doi: 10.1109/WKDD.2010.118.

Annika Meyer, Philipp Skudlik, Jan-Hendrik Pauls, and Christoph Stiller. Yolino: Generic single shot polyline detection in real time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2916–2925, 2021.

Himanshu Mittal, Avinash Chandra Pandey, Mukesh Saraswat, Sumit Kumar, Raju Pal, and Garv Modwel. A comprehensive survey of image segmentation: clustering methods, performance parameters, and benchmark datasets. *Multimedia Tools and Applications*, 81(24):35001–35026, October 2022. ISSN 1573-7721. doi: 10.1007/s11042-021-10594-9. URL <https://doi.org/10.1007/s11042-021-10594-9>.

Yujian Mo, Yan Wu, Xinneng Yang, Feilin Liu, and Yujun Liao. Review the state-of-the-art technologies of semantic segmentation based on deep learning. *Neurocomputing*, 493: 626–646, 2022. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2022.01.005>. URL <https://www.sciencedirect.com/science/article/pii/S0925231222000054>.

Bogdan Tomoyuki Nassu and Masato Ukai. Rail extraction for driver support in railways. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 83–88, 2011. doi: 10.1109/IVS.2011.5940410.

OpenCV. Structural analysis and shape descriptors, 2024a. URL https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga107a78bf7cd25dec05fb4dfc5c9e765f.

OpenCV. Fast line detector, 2024b. URL https://docs.opencv.org/4.x/df/ded/group_ximgproc_fast_line_detector.html.

I Pagand, C Carr, and J Doppelbauer. Fostering the railway sector through the european green deal. *European Union Agency For Railways*, 2020.

Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

Andrei I. Purica, Beatrice Pesquet-Popescu, and Frederic Dufaux. A railroad detection algorithm for infrastructure surveillance using enduring airborne systems. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2187–2191, 2017. doi: 10.1109/ICASSP.2017.7952544.

Zhiqian Qi, Yingjie Tian, and Yong Shi. Efficient railway tracks detection and turnouts recognition method using hog features. *Neural Computing and Applications*, 23: 245–254, 2013.

Ameet Annasaheb Rahane and Anbumani Subramanian. Measures of complexity for large scale image datasets. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)*, pages 282–287. IEEE, 2020.

RangeKing. Model structure of YOLOv8 detection models, 2023. URL <https://github.com/ultralytics/ultralytics/issues/189>.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

P.K Sahoo, S Soltani, and A.K.C Wong. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2):233–260, 1988. ISSN 0734-189X. doi: [https://doi.org/10.1016/0734-189X\(88\)90022-9](https://doi.org/10.1016/0734-189X(88)90022-9). URL <https://www.sciencedirect.com/science/article/pii/0734189X88900229>.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Adrian Straker, Stefano Puliti, Johannes Breidenbach, Christoph Kleinn, Grant Pearse, Rasmus Astrup, and Paul Magdon. Instance segmentation of individual tree crowns with yolov5: A comparison of approaches using the forinstance benchmark lidar dataset. *ISPRS Open Journal of Photogrammetry and Remote Sensing*, 9:100045, 2023. ISSN 2667-3932. doi: <https://doi.org/10.1016/j.photo.2023.100045>. URL <https://www.sciencedirect.com/science/article/pii/S2667393223000169>.

Rustam Tagiew, Martin Köppel, Karsten Schwalbe, Patrick Denzler, Philipp Neumaier, Tobias Klockau, Martin Boekhoff, Pavel Klasek, and Roman Tilly. Osdar23: Open sensor data for rail 2023. *arXiv preprint arXiv:2305.03001*, 2023.

Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, 15(1):29, 8 2015.

ISSN 1471-2342. doi: 10.1186/s12880-015-0068-x. URL <https://doi.org/10.1186/s12880-015-0068-x>.

Jigang Tang, Songbin Li, and Peng Liu. A review of lane detection methods based on deep learning. *Pattern Recognition*, 111:107623, 2021.

Zhu Teng, Feng Liu, and Baopeng Zhang. Visual railway detection by superpixel based intracellular decisions. *Multimedia Tools and Applications*, 75:2473–2486, 2016.

Tugce Toprak, Burak Belenlioglu, Burak Aydin, Cuneyt Guzelis, and M. Alper Selver. Conditional weighted ensemble of transferred models for camera based onboard pedestrian detection in railway driver support systems. *IEEE Transactions on Vehicular Technology*, 69(5):5041–5054, 2020. doi: 10.1109/TVT.2020.2983825.

TuSimple. Tusimple, 2017. URL https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection.

Ultralytics Inc. Yolov8, 2023. URL <https://docs.ultralytics.com/models/yolov8/>.

Jinsheng Wang, Yinchao Ma, Shaofei Huang, Tianrui Hui, Fei Wang, Chen Qian, and Tianzhu Zhang. A keypoint-based global association network for lane detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1392–1401, 2022.

Yin Wang, Lide Wang, Yu Hen Hu, and Ji Qiu. Railnet: A segmentation network for railroad detection. *IEEE Access*, 7:143772–143779, 2019. doi: 10.1109/ACCESS.2019.2945633.

Bisheng Yang and Lina Fang. Automated extraction of 3-d railway tracks from mobile laser scanning point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(12):4750–4761, 2014. doi: 10.1109/JSTARS.2014.2312378.

Mingyue Yang. Lane detection methods survey for automatic driving. In *Journal of Physics: Conference Series*, volume 2547, page 012015. IOP Publishing, 2023.

Xiang Yue, Kai Qi, Xinyi Na, Yang Zhang, Yanhua Liu, and Cuihong Liu. Improved yolov8-seg network for instance segmentation of healthy and diseased tomato plants in the growth stage. *Agriculture*, 13(8), 2023. ISSN 2077-0472. doi: 10.3390/agriculture13081643. URL <https://www.mdpi.com/2077-0472/13/8/1643>.

Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoonah Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, 126:103514, 2022. ISSN 1051-2004. doi: <https://doi.org/10.1016/j.dsp.2022.103514>. URL <https://www.sciencedirect.com/science/article/pii/S1051200422001312>.

Oliver Zendel, Markus Murschitz, Marcel Zeilinger, Daniel Steininger, Sara Abbasi, and Csaba Beleznai. Railsem19: A dataset for semantic rail scene understanding. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1221–1229, 2019. doi: 10.1109/CVPRW.2019.00161.

Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. Clrnet: Cross layer refinement network for lane detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 898–907, 2022.

List of Figures

Figure 1	Locations where images were captured around Hamburg, Germany.	6
Figure 2	Example images of high resolution RGB, low resolution RGB, and infrared sensors (Tagiew et al., 2023).	7
Figure 3	Number of images and labels per sensor, respectively.	8
Figure 4	Track labels per image. Most images depict track pairs. However, there are also images with an odd number of tracks.	8
Figure 5	Brightness of images by sensor.	9
Figure 6	Examples of very bright and very dark images, respectively.	10
Figure 7	Entropy of images by sensor.	11
Figure 8	Examples of images with minimal and maximal entropy, respectively.	11
Figure 9	Histogram showing the occlusion level for track labels.	12
Figure 10	Examples of track labels with 100% occlusion.	12
Figure 11	Three examples with seven video frames (i.e., images), respectively.	13
Figure 12	Track labels per image on a logarithmic scale. The images range from simple railroads with a single pair of tracks to complicated networks with 26 pairs of tracks.	14
Figure 13	Examples of simple and complicated rail infrastructures.	14
Figure 14	Brightness and entropy of RailSem19 images.	15
Figure 15	Extreme examples with regard to brightness and entropy.	16
Figure 16	Process of transforming the labels from polyline to polygon.	19
Figure 17	Two different approaches for labeling tracks.	19
Figure 18	Example masks – ground truth vs. prediction.	22
Figure 19	Different bounding box aspect ratios depending on camera orientation.	23
Figure 20	Histogram of bounding box aspect ratios by camera orientation.	24
Figure 21	Baselining approach based on fast line detection and post-processing.	25
Figure 22	Models with different train and validation sets.	30
Figure 23	Comparison between YOLOv8 models with unified labels and separate labels by sensor type.	34
Figure 24	Example results of different labeling approaches.	34

Figure 25 Comparison between YOLOv8 (separate labels) and FLD based on Dice score. Sensor types are distinguished by color.	35
Figure 26 Example image where FLD performs better than YOLOv8; predictions are colored red.	36
Figure 27 Example where FLD slightly outperforms YOLOv8.	36
Figure 28 Extreme example where YOLOv8 clearly outperforms FLD.	37
Figure 29 Dice scores vs. entropy scores – FLD and YOLOv8 are distinguished by color.	37
Figure 30 Dice scores vs. brightness scores – FLD and YOLOv8 are distinguished by color.	38
Figure 31 Violin plot indicating the difference between YOLOv8 and FLD on sensor level.	38
Figure 32 Converting masks to polylines.	39
Figure 33 Illustration of “rail track detection” algorithm.	41
Figure 34 OSDaR23 data split on image level in total and on sensor level.	56
Figure 35 Heatmap of Dice scores for selected parameter pairs. The Dice scores are average values of 196 images.	57
Figure 36 Model structure of YOLOv8 detection models.	58
Figure 37 Training results of data_highres_railsem_640 model with unified labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.	59
Figure 38 Example of training and validation batches during training model data_highres_railsem_640 with unified labels.	59
Figure 39 Training results of data_all_640 model with unified labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.	60
Figure 40 Example of training and validation batches during training model data_all_640 with unified labels.	60
Figure 41 Training results of data_highres_railsem_640 model with separate labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.	61
Figure 42 Example of training and validation batches during training model data_highres_railsem_640 with separate labels.	61
Figure 43 Training results of data_al_railsem_640 model with separate labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.	62

Figure 44 Example of training and validation batches during training model data_all_railsem_640 with separate labels.	62
Figure 45 Parameter tuning of model data_highres_railsem_640 with AdamW optimizer.	63
Figure 46 Parameter tuning of model data_highres_railsem_640 with SGD opti- mizer.	64
Figure 47 Parameter tuning of model data_all_640 with AdamW optimizer.	64
Figure 48 Parameter tuning of model data_all_640 with SGD optimizer.	65

List of Tables

Table 1	Size of images per sensor.	8
Table 2	Sensor parameters.	26
Table 3	Best parameter settings and associated performance metrics.	26
Table 4	Training details.	28
Table 5	Inference speed in milliseconds.	30
Table 6	Parameter tuning with unified labels.	31
Table 7	Comparison of automatically chosen parameters and tuned parameters based on Dice score with unified labels.	32
Table 8	OSDaR23 sensor specifications.	54
Table 9	Dataset split: OSDaR23.	55
Table 10	Bounding box aspect ratios for different sensors; Q* denotes quantile.	56
Table 11	Description of YOLO parameters.	63

A Appendix

A.1 Sensors used in OSDaR23

In this thesis, we are focusing on images from the different sensor types.

Three 12MP RGB Cameras	
Type	Teledyne GenieNano 5GigE C4040
Sensor data	RGB images (8 Bit, PNG)
Resolution	4,112 × 2,504 px
Sampling frequency	10 Hz (synchronized)
Alignment	trident (in driving direction diagonal left, central and diagonal right)

Three 5MP RGB Cameras	
Type	Teledyne GenieNano C2420
Sensor data	RGB images (8 Bit, PNG)
Resolution	2,464 × 1,600 px
Sampling frequency	10 Hz (synchronized)
Alignment	trident

Three IR Cameras	
Type	Teledyne Calibir DXM640
Sensor data	Grayscale images (8 Bit, PNG)
Resolution	640 × 480 px
Sampling frequency	10 Hz (synchronized)
Alignment	trident

Table 8: OSDaR23 sensor specifications.

A.2 Dataset split

Splitting the OSDaR23 dataset into train, validation, and test subsets is performed based on video sequences. Table 9 lists the assignment of sequences to the respective subset. Figure 34 shows the split on an image level in total and for each sensor type.

Train	Validation	Test
10_station_sueldorf_10.1	16_under_bridge_16.1	14_signals_station_14.1
11_main_station_11.1	18_vegetation_switch_18.1	21_station_wedel_21.1
12_vegetation_steady_12.1	1_calibration_1.2	4_station_pedestrian_bridge_4.2
13_station_ohlsdorf_13.1	3_fire_site_3.3	4_station_pedestrian_bridge_4.5
14_signals_station_14.2	9_station_ruebenkamp_9.1	7_approach_underground_station_7.2
14_signals_station_14.3	9_station_ruebenkamp_9.2	8_station_altona_8.3
15_construction_vehicle_15.1		
17_signal_bridge_17.1		
19_vegetation_curve_19.1		
1_calibration_1.1		
20_vegetation_squirrel_20.1		
21_station_wedel_21.2		
21_station_wedel_21.3		
2_station_berliner_tor_2.1		
3_fire_site_3.1		
3_fire_site_3.2		
3_fire_site_3.4		
4_station_pedestrian_bridge_4.1		
4_station_pedestrian_bridge_4.3		
4_station_pedestrian_bridge_4.4		
5_station_bergedorf_5.1		
5_station_bergedorf_5.2		
6_station_klein_frottbek_6.1		
6_station_klein_frottbek_6.2		
7_approach_underground_station_7.1		
7_approach_underground_station_7.3		
8_station_altona_8.1		
8_station_altona_8.2		
9_station_ruebenkamp_9.3		
9_station_ruebenkamp_9.4		
9_station_ruebenkamp_9.5		
9_station_ruebenkamp_9.6		
9_station_ruebenkamp_9.7		

Table 9: Dataset split: OSDaR23.

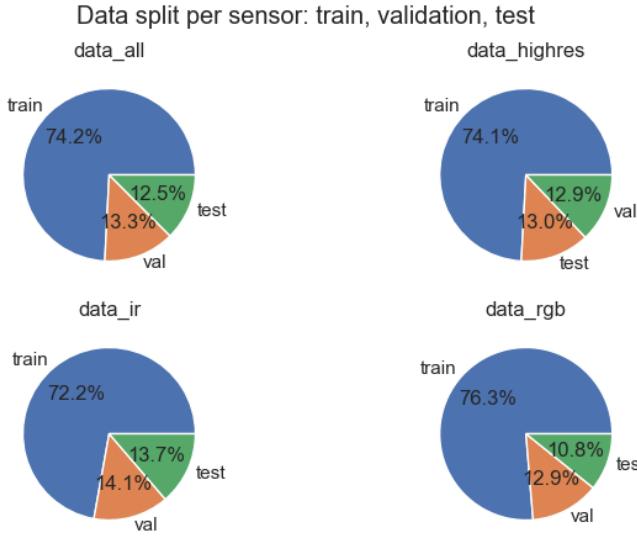


Figure 34: OSDaR23 data split on image level in total and on sensor level.

A.3 Fast line detection

Table 10 gives the statistics of the aspect ratios of bounding boxes enclosing track labels.

Sensor	Mean	Median	Q1	Q5	Q95	Q10	Q90	Q99
ir_center	0.714	0.379	0.104	0.113	2.210	0.126	1.857	3.917
ir_left	1.215	1.082	0.121	0.344	2.315	0.451	2.293	2.591
ir_right	1.849	1.872	0.423	0.813	3.399	1.000	2.688	4.831
rgb_center	1.283	1.057	0.127	0.208	3.542	0.232	3.085	5.487
rgb_highres_center	0.671	0.618	0.066	0.071	1.804	0.073	1.561	2.620
rgb_highres_left	1.320	1.120	0.138	0.654	2.192	0.738	2.181	2.578
rgb_highres_right	1.623	1.191	0.169	0.607	3.627	0.690	3.580	4.324
rgb_left	2.151	1.815	0.353	0.700	3.882	0.873	3.845	5.632
rgb_right	2.522	1.890	0.143	0.951	5.515	1.329	4.101	8.476

Table 10: Bounding box aspect ratios for different sensors; Q* denotes quantile.

Figure 35 shows the effect of different parameters on the performance measured by the Dice score. The results are averaged over 196 images.

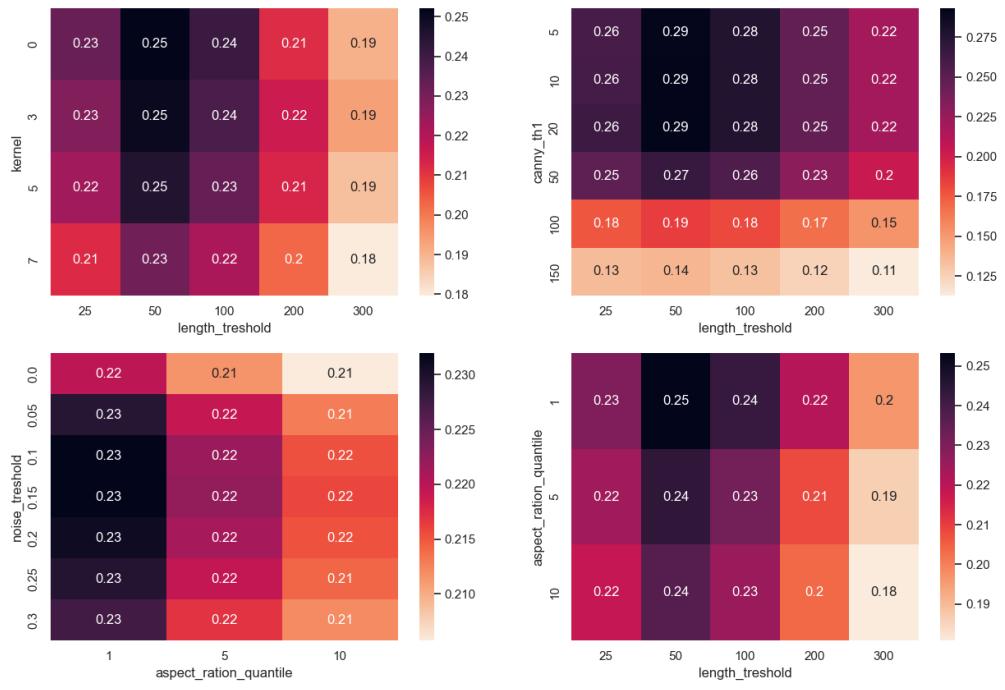


Figure 35: Heatmap of Dice scores for selected parameter pairs. The Dice scores are average values of 196 images.

A.4 YOLOv8

Figure 36 presents the model architecture of YOLOv8 (RangeKing, 2023).

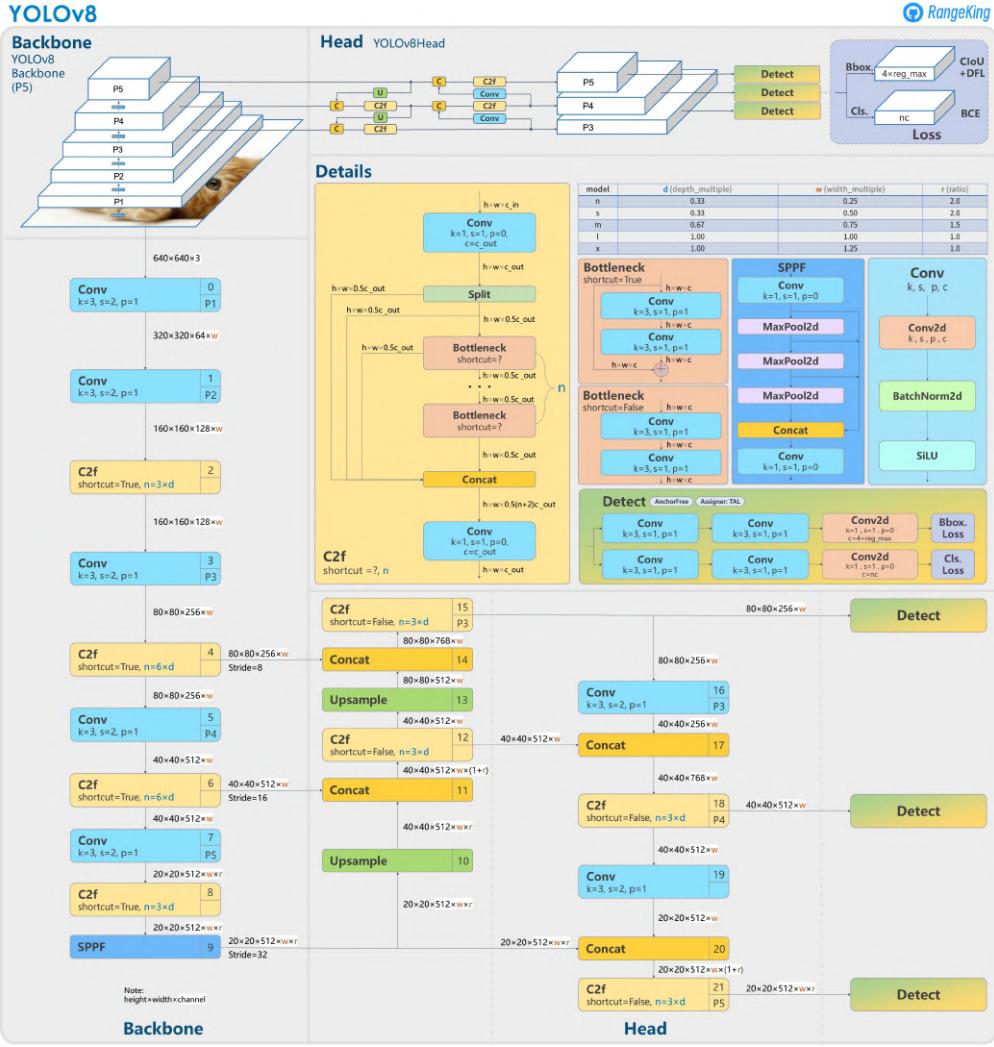


Figure 36: Model structure of YOLOv8 detection models.

A.4.1 Results of data_highres_railsem_640 model with unified labels

Figure 37 shows how the performance indicators evolve during the training process, while Figure 38 gives an example of a training batch and a validation batch, respectively. Figure 38a demonstrates the mosaic data augmentation that was introduced in YOLOv8. Mosaic data augmentation is a technique where four different images are mixed together and are then provided to the model as input. This technique allows the model to learn the actual objects from different positions and with partial occlusion.

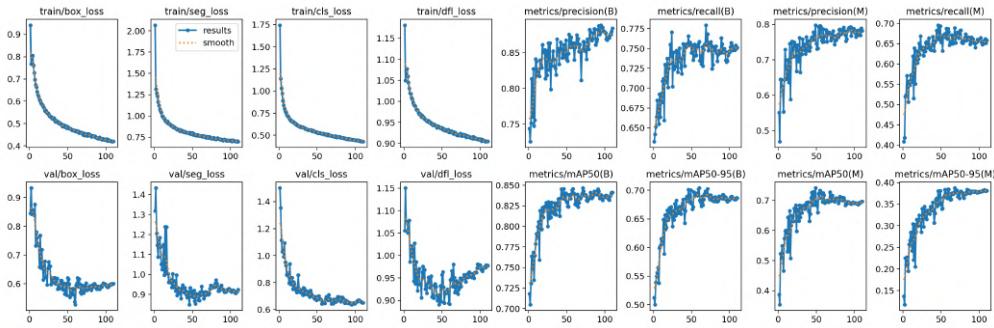
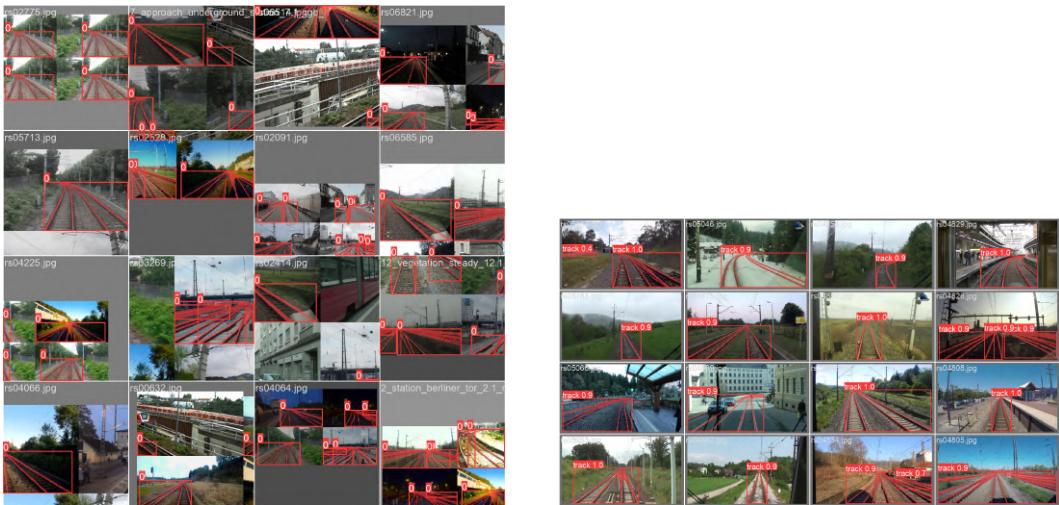


Figure 37: Training results of data_highres_railsem_640 model with unified labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.



(a) Example of training batch.
Images are augmented.

(b) Example of validation batch.

Figure 38: Example of training and validation batches during training model data_highres_railsem_640 with unified labels.

A.4.2 Results of data_all_640 model with unified labels

Figure 39 shows how the performance indicators evolve during the training process, while Figure 40 gives an example of a training batch and a validation batch, respectively. Figure 40b demonstrates the downside of having many similar images that are taken from one video sequence – the validation set is very restricted and therefore not always representative to the wide range of possible scenarios.

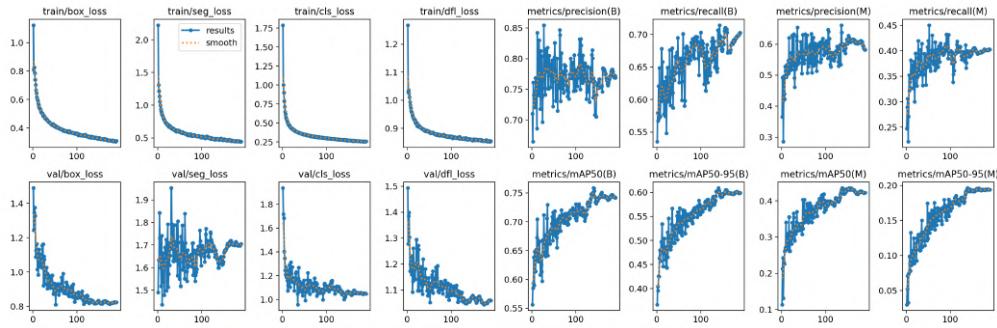
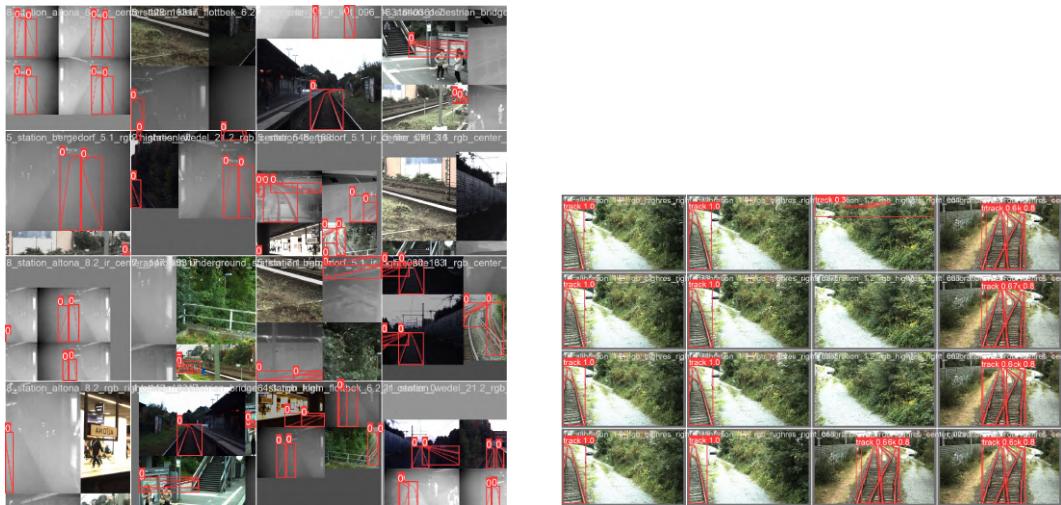


Figure 39: Training results of data_all_640 model with unified labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.



(a) Example of training batch.

Images are augmented.

(b) Example of validation batch.

Images are frames of video sequence.

Figure 40: Example of training and validation batches during training model data_all_640 with unified labels.

A.4.3 Results of data_highres_railsem_640 model with separate labels

Figure 41 shows how the performance indicators evolve during the training process, while Figure 42 gives an example of a training batch and a validation batch, respectively.

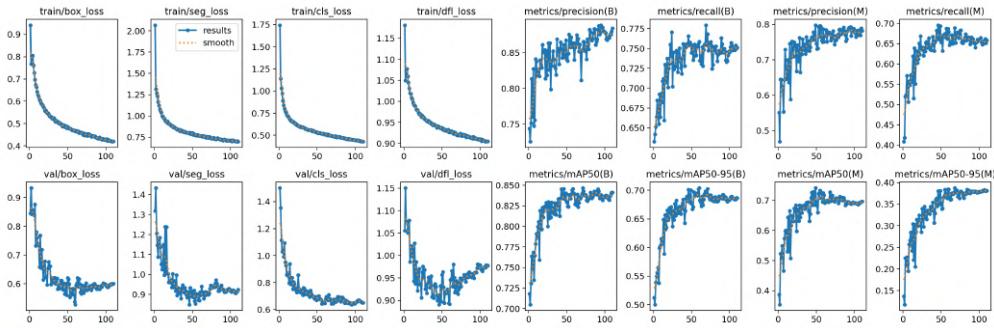
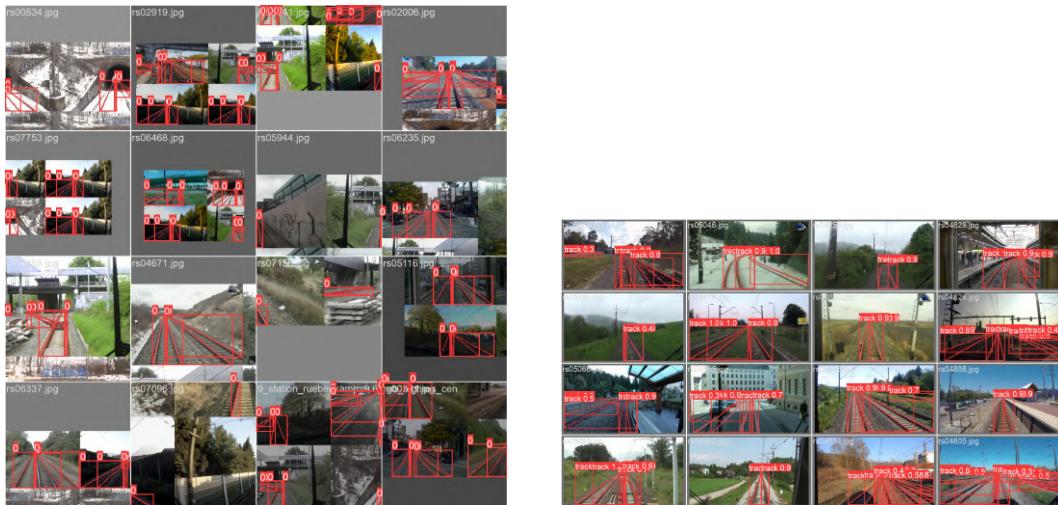


Figure 41: Training results of data_highres_railsem_640 model with separate labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.



(a) Example of training batch.

Images are augmented.

(b) Example of validation batch.

Figure 42: Example of training and validation batches during training model data_highres_railsem_640 with separate labels.

A.4.4 Results of data_all_railsem_640 model with separate labels

Figure 43 shows how the performance indicators evolve during the training process, while Figure 44 gives an example of a training batch and a validation batch, respectively.

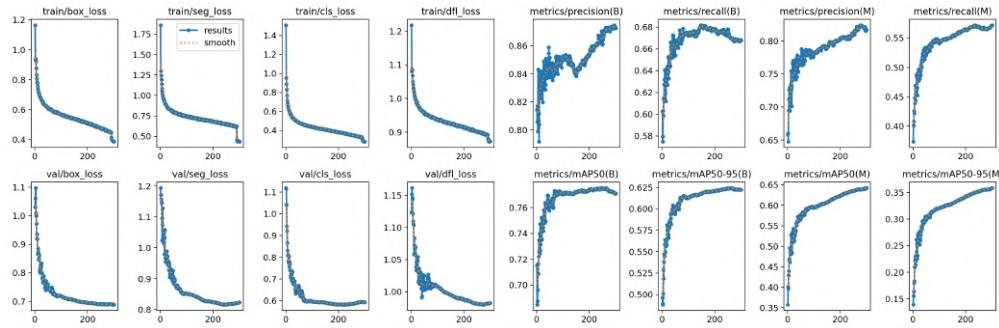
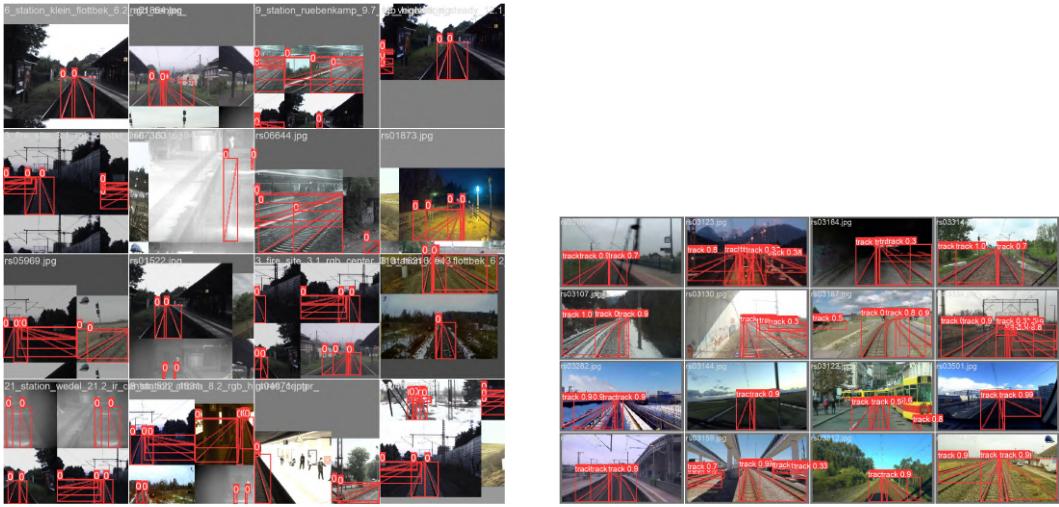


Figure 43: Training results of data_al_railsem_640 model with separate labels. (B) stands for the results of the bounding box and (M) stands for the result of the segmentation mask.



(a) Example of training batch.

Images are augmented.

(b) Example of validation batch.

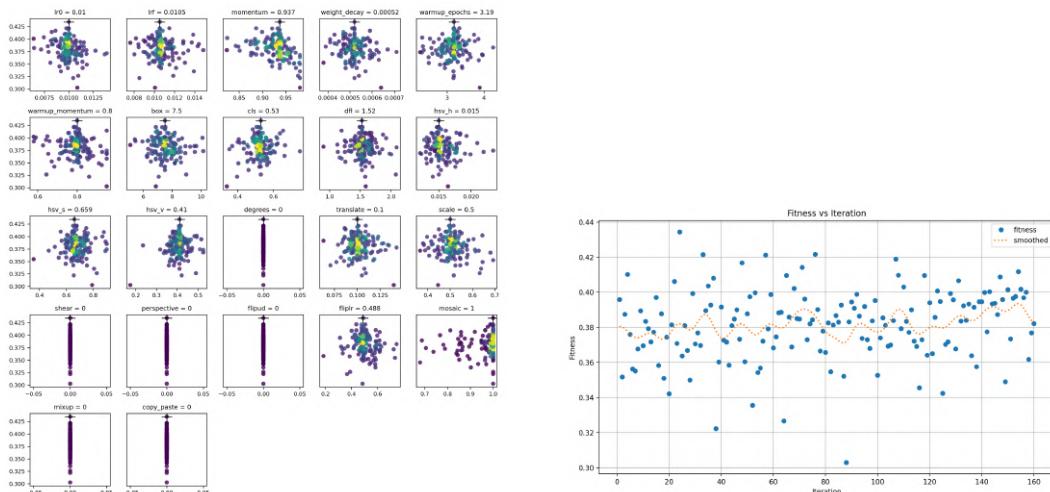
Figure 44: Example of training and validation batches during training model data_all_railsem_640 with separate labels.

A.4.5 Parameter tuning with unified labels

Figure 45 and Figure 46 visualize the tuning process on the data_highres_railsem_640 model with AdamW and SGD optimizer, respectively. The parameters are described in Table 11. Certain parameters, such as degrees, translate, etc. are fixed at zero throughout the tuning process as it is unlikely that the resulting images appear in the real-world.

Parameter	Description
lr0	initial learning rate
lrf	final learning rate
momentum	SGD momentum/Adam beta1
weight_decay	optimizer weight decay
warmup_momentum	warmup initial momentum
box	box loss gain
cls	classification loss gain
dfl	dual focal loss gain
hsv_s	image Hue Saturation Value augmentation (fraction)
hsv_v	image Hue Saturation Value augmentation (fraction)
degrees	image rotation (+/- deg)
translate	image translation (+/- fraction)
shear	image shear (+/- deg)
perspective	image perspective (+/- fraction),
flipud	image flip up-down (probability)
filpr	image flip left-right (probability)
mixup	image mixup (probability)
copy_paste	segment copy-paste (probability)

Table 11: Description of YOLO parameters.



- (a) Search space for different parameters. Density is distinguished by color: yellow shades indicate higher density.
- (b) Evolution of fitness value over 160 iterations. Model fitness is a weighted combination of different metrics (e.g., precision, recall, mAP@0.5).

Figure 45: Parameter tuning of model data_highres_railsem_640 with AdamW optimizer.

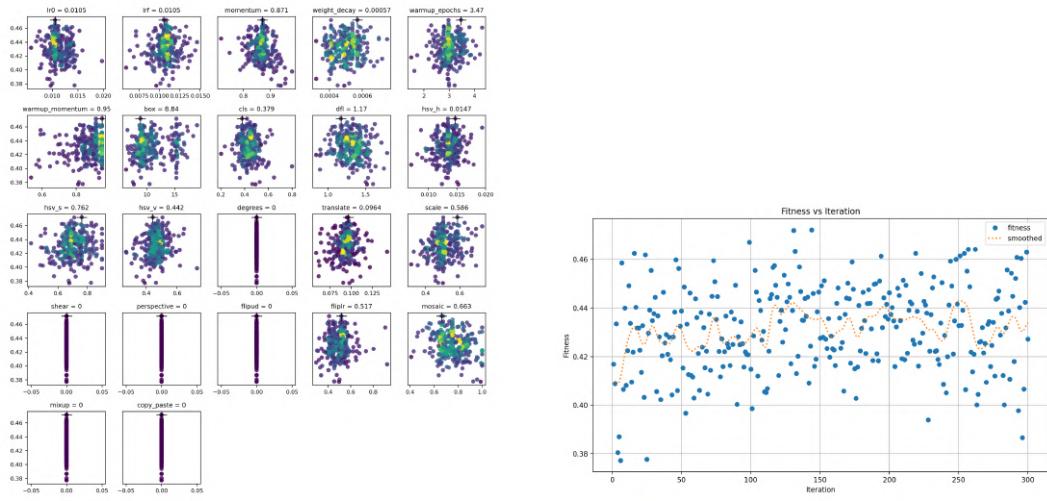


Figure 46: Parameter tuning of model data_highres_railsem_640 with SGD optimizer.

Figure 47 and Figure 48 visualize the tuning process on the data_all_640 model with AdamW and SGD optimizer, respectively.

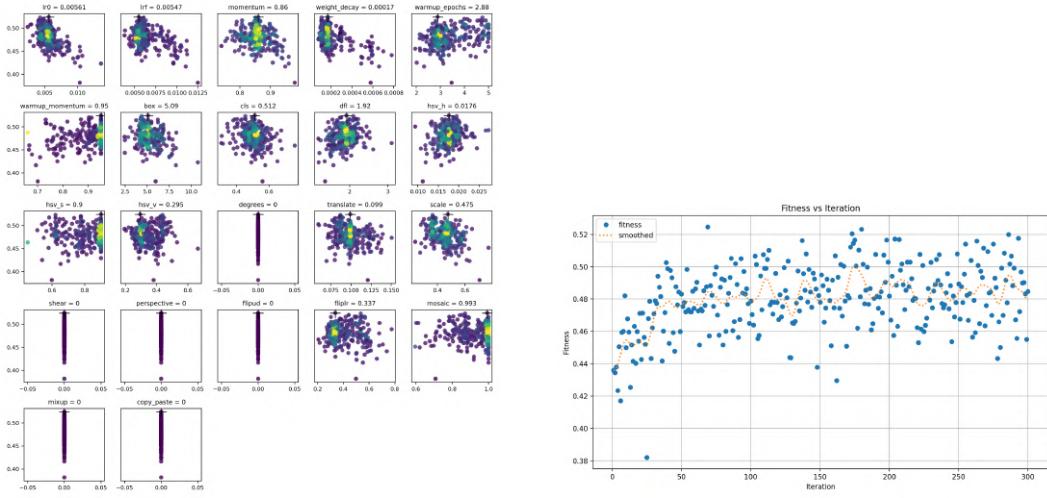


Figure 47: Parameter tuning of model data_all_640 with AdamW optimizer.

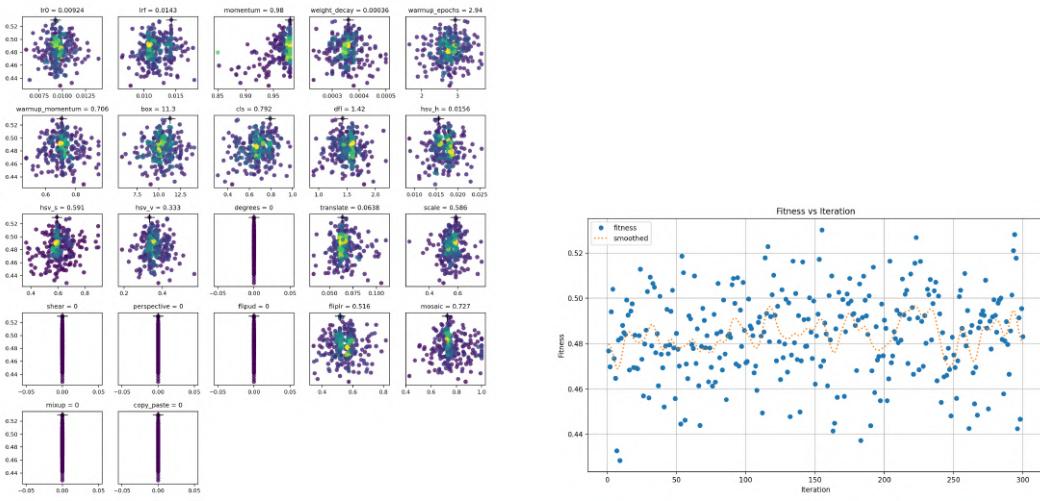


Figure 48: Parameter tuning of model data_all_640 with SGD optimizer.