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 Mechatronics/Robotics

Rail Track Prediction
Arbeitstitel

By: Sebastian Goebel, BSc.

Student Number: University of Applied Science: 51912403
Luzern University of Applied Sciences and Arts: 23-565-161

Supervisor: Prof. Dr. Björn Jensen

Vienna, November 21, 2024

Christian Doppler Forschungsgesellschaft

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 Urheberrechtsgesetz /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.“

Vienna, November 21, 2024 Signature

Kurzfassung

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Schlagworte: Schlagwort1, Schlagwort2, Schlagwort3, Schlagwort4

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Keywords: Keyword1, Keyword2, Keyword3, Keyword4

Acknowledgements

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1 Erste Überschrift der Tiefe 1 (chapter) 1
1.1 Erste Überschrift Tiefe 2 (section) . . . 1
1.1.1 Erste Überschrift Tiefe 3 (subsection) . . . 1

2 Zweite Überschrift der Tiefe 1 (chapter) 1
2.1 Zweite Überschrift Tiefe 2 (section) . . . 1
2.1.1 Zweite Überschrift Tiefe 3 (subsection) . . . 1
2.1.2 Dritte Überschrift Tiefe 3 (subsection) . . . 1

3 Dritte Überschrift der Tiefe 1 (chapter) 3
3.1 Algorithms . . . 3

4 Introduction 4

5 State of the Art 7
5.1 Different approaches of vision tasks . . . 7
5.2 Real-time Object detection . . . 9
5.2.1 Two-stage detectors . . . 9
5.2.2 One-stage detectors . . . 9
5.3 Semantic Segmentation . . . 11
5.3.1 Example Überschrift . . . 13
5.3.2 Example Überschrift . . . 13
5.3.3 Rail Segmentation oder Semantic Segmentation (2 bis 3 Seiten) . . . 13
5.3.4 Line detection algorithms (2 bis 3 Seiten) . . . 14
5.3.5 Backbone architectures 2 Seiten . . . 14
5.3.6 mögliches Baseline Paper vielleicht (wenn die Aufteilung geändert werden muss) . . . 14
5.3.7 verwendetes Baseline Paper vielleicht (wenn die Aufteilung geändert werden muss) . . . 14
5.3.8 Temporal Models (2 Seiten) . . . 14
5.4 Network architectures . . . 15
5.4.1 ResNets . . . 16
5.4.2 DenseNets . . . 16
5.4.3 MobileNetV3 . . . 17
5.4.4 EfficientNets . . . 18

5.5 Datasets . . . 20
5.6 Baseline Paper . . . 32
5.6.1 Annotations . . . 32
5.6.2 Data augmentation . . . 32
5.6.3 TEP-Net Model . . . 34
5.6.4 Loss function . . . 36
5.6.5 Experiments, Results and Comparison other state-of-the-art approaches of TEP-Net . . . 38
5.6.6 Limitation . . . 40

6 Methodology **42**
6.1 Datasets . . . 42
6.1.1 RailSem19 and used subsets . . . 42
6.1.2 vielleicht: Rail-DB for multi rail setection . . . 42
6.1.3 used annotations . . . 42
6.2 Switch evaluation dataset . . . 42
6.3 labeling task for temporal models . . . 42
6.4 Setup used for Training CNNs . . . 43
6.5 Measuing Inference on NVIDIA Jetson Device . . . 43
6.5.1 Hardware Setup for Measuring Inference . . . 43
6.5.2 Optimizing models with TensorRT . . . 44

7 Experiments **47**
7.1 Ensemble Learning Approach . . . 47
7.2 improved TEP-Net . . . 47
7.2.1 Autocrop . . . 47
7.2.2 Backbones . . . 47
7.2.3 Pooling Layers . . . 47
7.2.4 Prediction Heads . . . 47
7.2.5 Sliding Window Approach . . . 47
7.2.6 Temporal Models . . . 47
7.2.7 Erste Überschrift Tiefe 3 (subsection) . . . 48
7.2.8 Erste Überschrift Tiefe 3 (subsection) . . . 48

8 Results **49**
8.1 Ensemble Learning Approach . . . 49
8.2 improved TEP-Net . . . 49
8.2.1 Autocrop . . . 49
8.2.2 Backbones . . . 49
8.2.3 Pooling Layers . . . 49
8.2.4 Prediction Heads . . . 49

8.2.5 Sliding Window Approach 49
8.2.6 Temporal Models 49
8.3 Erste Überschrift Tiefe 2 (section) 50
8.3.1 Erste Überschrift Tiefe 3 (subsection) 50

9 Discussion **51**
9.1 Erste Überschrift Tiefe 2 (section) 51
9.1.1 Erste Überschrift Tiefe 3 (subsection) 51

10 Conclusion and Outlook **52**
10.1 Erste Überschrift Tiefe 2 (section) 52
10.1.1 Erste Überschrift Tiefe 3 (subsection) 52

Bibliography **53**

List of Figures **62**

List of Tables **64**

Quellcodeverzeichnis **65**

Abkürzungsverzeichnis **66**

A Anhang A **68**

B Anhang B **69**

1 Erste Überschrift der Tiefe 1 (chapter)

Etwas Text... Hier kommen noch einige Abkürzunge vor zum Beispiel Alphabet (ABC),world wide web (WWW) und Rolling on floor laughing (ROFL).

1.1 Erste Überschrift Tiefe 2 (section)

blindtext

1.1.1 Erste Überschrift Tiefe 3 (subsection)

blindtext

Erste Überschrift Tiefe 4 (subsubsection)

blindtext

2 Zweite Überschrift der Tiefe 1 (chapter)

blindtext

2.1 Zweite Überschrift Tiefe 2 (section)

blindtext

2.1.1 Zweite Überschrift Tiefe 3 (subsection)

blindtext

2.1.2 Dritte Überschrift Tiefe 3 (subsection)

blindtext

Zweite Überschrift Tiefe 4 (subsubsection)

blindtext
Querverweise werden in LaTeX automatisch erzeugt und verwaltet, damit sie leicht aktualisiert werden können. Hier wird zum Beispiel auf Abbildung 1 verwiesen.

Figure 1: Beispiel für die Beschriftung eines Buchrückens.

Figure 2: 2. Beispiel für die Beschriftung eines Buchrückens.

Und hier ist ein Verweis auf Tabelle 1. Das gezeigte Tabellenformat ist nur ein Beispiel. Tabellen können individuell gestaltet werden.

Table 1: Semesterplan der Lehrveranstaltung „Angewandte Mathematik“.

Datum	Thema	Raum
20.08.2008	Graphentheorie	HS 3.13
01.10.2008	Biomathematik	HS 1.05

Table 2: 2. Semesterplan der Lehrveranstaltung „Angewandte Mathematik“.

Datum	Thema	Raum
20.08.2008	Graphentheorie	HS 3.13
01.10.2008	Biomathematik	HS 1.05

Hier wird auf die Formel 1 verwiesen.

$$x = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q} \tag{1}$$

$$x = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q} \tag{2}$$

Literaturverweise sollten automatisch verwaltet werden, vor allem, wenn es viele Quellenverweise gibt. Beispiele sind [1] [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]. Das verwendete Zitierformat (bzw. das Format des Literaturverzeichnisses) ist entspechend der Vorgaben der Studiengänge zu wählen.

3 Dritte Überschrift der Tiefe 1 (chapter)

Hier wird etwas Quellcode dargestellt:

```
#include <iostream>

void SayHello(void)
{
    // Kommentar
    cout << "Hello World!" << endl;
}

int main(int argc, char **argv)
{
    SayHello();
    return 0;
}
```

Listing 1: Hello-World

3.1 Algorithms

Use a defined environment for algorithms.

Algorithm 1 is an example from the gallery (https://www.overleaf.com/latex/examples/euclids-algorithm-an-example-of-how-to-write-algorithms-in-latex/mbysznrmktqf) .

Algorithm 1 Euclid's algorithm

procedure EUCLID(a, b) ▷ The g.c.d. of a and b
  $r \leftarrow a \bmod b$
  while $r \neq 0$ **do** ▷ We have the answer if r is 0
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a \bmod b$
  return b ▷ The gcd is b

4 Introduction

After a drop in rail passenger transport in 2020, it quickly recovered and increased by almost 71%, resulting in 328,3 million passengers in 2023 [16]. The Wiener Linien and the ÖBB show similar behaviors with an increase of 218 million and 115,4 million passengers from 2020 to 2023 respectively [17] [18].

The increase in train traffic is bound to a rise in accidents. In Austria, 34 rail traffic accidents with 38 victims were recorded in 2021. Of them 24 were seriously injured and 14 were killed [19]. Incidents include a train derailment in the area of a signal box, where there usually are many switches [20]. In 2022 in Münchendorf a train drove too fast over a switch, resulting in a derailment accident, with one passenger dead and 25 injured [21]. Also in other countries like the Czech Republic, an accident occurred, where a head-on collision of two trains resulted in four deaths. It is suspected that an incorrect switch caused that incident [22]. The cause of all accidents mentioned can be traced back to either an incorrect set switch or a derailment at a switch. This leads to the idea that correctly identifying the switch states may have prevented these accidents.

(a) Accident in Münchendorf, Austria. The train drove too fast over a switch and derailed, resulting in one death and 25 injured. [21]

(b) Accident in the Czech Republic. Head-on collision of two trains. The suspected cause is an incorrecty set switch, resulting in four deaths. [22]

Figure 3: Train accidets in **(a)** Austria and **(b)** the Czech Republic, where misinformation of switches resulted in deaths.

Simultaneously, autonomous vehicles are becoming increasingly seen as revolutionary technology for the future [23]. Also indicated by the "Autonomous Vehicle Readiness Index" [24] in most countries including Austria [25]. Especially autonomous road vehicles are rapidly evolving and could improve safety [26] [1], with machine-learning algorithms being a key trend [26]. The rail domain on the other hand, has gotten little attention [26]. This is because, unlike other

modes of transportation like airplanes or cars, trains follow a predetermined path defined by the tracks. Nevertheless, trams or other low-speed trains operate in dynamic environments where autonomous systems are still needed to respond to unpredictable events [1]. Therefore autonomy for these modes of transport will gain importance in the future [27].

One particular task of an autonomous train system involves filtering out the area in front of the vehicle, which can be defined as the danger zone. This area represents where the train is headed and the space, which is occupied next.

The problem is, that many state-of-the-art techniques in the rail domain consider all rails or rail tracks, and often make no distinction between the train's track and other tracks [1]. This can be observed in figure 10, in the right two images of figure 11 and in figure 12. While the segmentation of rails can be used for obstacle detection [27], the output of these methods can easily be misinterpreted [1]. Train infrastructure is complex, with multiple tracks often crossing, merging, or splitting at switches. Therefore, it is insufficient to recognize all tracks in a single image. A more targeted system is needed, which only detects the train's track and predicts the upcoming path at switches [1]. Such a system identifies the most relevant danger zone and when it is real-time capable it could be used as a preprocessing step for other algorithms or safety features [1] [27]. Leading to better-informed decisions [1], possibly preventing accidents like the ones mentioned before and potentially save lifes. Figure 4 shows a schematic visualization of a more targeted system.

This is why in the line of this work a distinction between *rail detection* and *rail track prediction* is made. On the one hand, *rail detection* in this context is filtering out rails or rail tracks with no further information about the own or adjacent rails. On the other hand, *rail track prediction* answers the question of where the train will be in the next few moments, particularly when switches are present that split the track. In figure 4 a *rail detection* would ouput both the green and the red marked tracks as one class and a *rail track prediction* outputs just the green track.

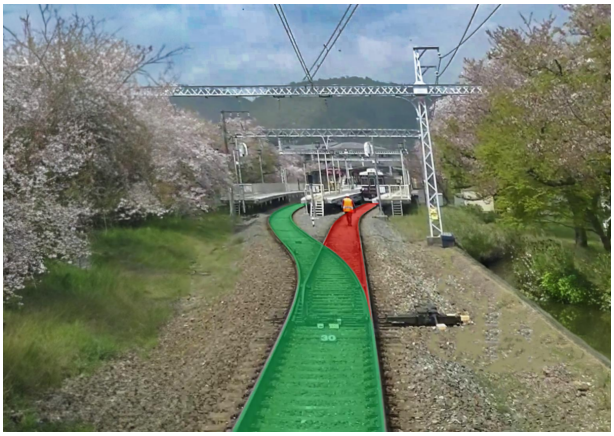


Figure 4: Example of a diverging rail track at a switch. The trains path is marked in green. The path the train cannot take is marked in red. This path is be obstructed and would be unsafe [1]. *rail detection*-output: green and red track; *rail track prediction*-output: green track

The focus of *rail track prediction* extends beyond solely camera-based detection of the train's path. It particularly emphasizes the accurate identification of switches where the train's path diverges, as illustrated in Figure 4.

Consequently, the goal of this work is the development of a rail track prediction system, which correctly predicts the most relevant danger zone and the direction in which the train is moving This is done with an emphasis on scenarios involving track splits through switches present in the Field Of View (FOV). Since the use case primarily takes place in dynamic environments, another goal, besides achieving an acceptable accuracy, is to ensure it operates in real-time and is therefore lightweight [1].

As the goal is to perceive the environment in front of the train, it is most advantageous to use front-view cameras [1] [27]. Images are captured in the driving direction from the driver's cabin because it is the best view of the tracks [1].

The main contributions of this work can be roughly divided into three parts.

1. Firstly, an initial approach was taken that aimed to combine object detection and semantic segmentation. However, due to unsatisfactory results, this methodology was deemed ineffective, leading to the pursuit of a fundamentally different solution.

2. Secondly, TEP-Net [1] was chosen as the new baseline for further experiments. Improvements are implemented with a specific focus on correct path prediction in the presence of switches.

3. Thirdly, as further improvements were anticipated through the use of the temporal component with RNNs, the system was adjusted to accept video information rather than single frames as input. Thus, the data loading logic was adapted and models were modified. Additionally, a temporal dataset was created. To minimize the workload for annotation, an auto labeler was developed utilizing the improved single-frame-based model.

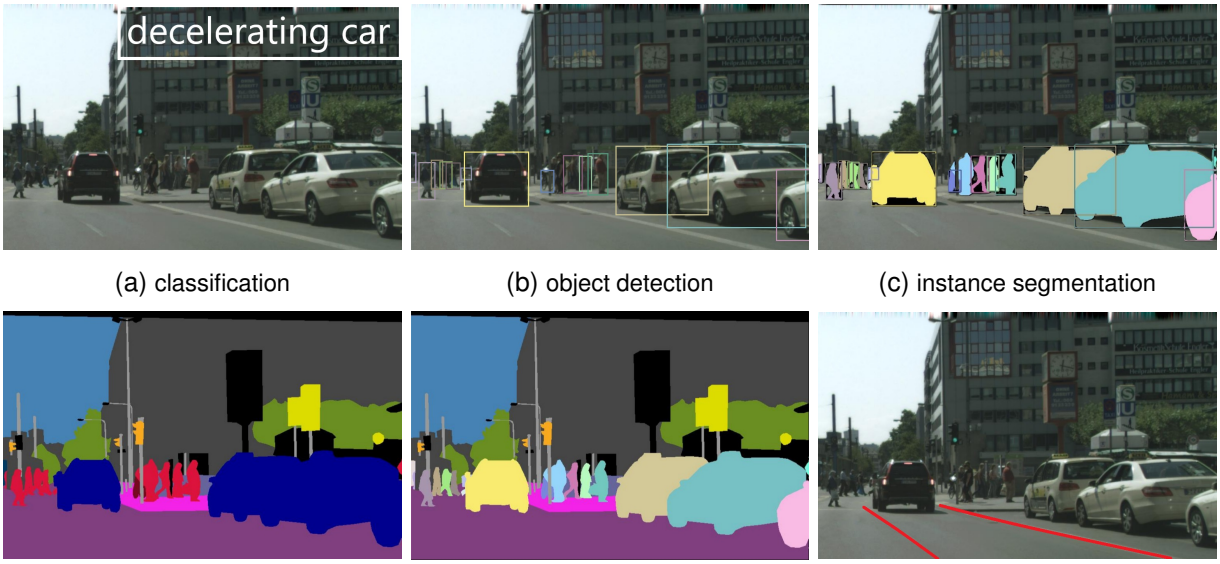
5 State of the Art describes different approaches, to how the train track prediction problem can be solved. In this section, thorough research is done, which gives an overview of various approaches, models, and fitting datasets. Additionally, a couple of papers that could serve as a baseline are discussed in more detail. In **6 Methodology** the used datasets as well as the hardware and software frameworks used for training and evaluation are described. Additionally, this work includes the development of a temporal dataset with partly auto-labeled annotations. The labeling process and algorithms for auto-labeling are also described in this section. After that carried-out experiments are described in **7 Experiments** and their results are described in **8 Results** following a discussion in **9 Discussion**. In the final section **10 Conclusion and Outlook**, the work is summarized and further possible ideas for improvements are presented.

5 State of the Art

To get an idea of possible solutions and the most promising approaches. There are many different kinds of vision tasks based on image input. This chapter introduces to different vision tasks and their approaches to serve as an overview. Then, the most promising approaches for the rail track prediction use case are selected and further pursued for comparison.

5.1 Different approaches of vision tasks

Many different kinds of vision tasks input image data. These use cases outline broad guidelines for designing neural network architectures and the workflow of such a model by which the desired output is produced. Figure 5 shows the outputs of the most commonly used strategies in the literature that might be relevant for rail track prediction or a part of it. The input image for all models is visualized in Figure 5a without the added text in the top right corner.



(a) classification (b) object detection (c) instance segmentation

(d) semantic segmentation (e) panoptic segmentation (f) line detection

Figure 5: The most common applications in vision tasks that are supported by neural networks. Shown are Ground Truth (GT) examples for each usage domain. The input image is visualized in (a) without the added text in the top right corner [28].

Classification describes the scene in the input image with only one label. Figure 5a shows a possible example with the label added in the top right corner of the input image. No further information is available from the output of a classification method.

Object detection combines classification with localization. This technique outputs so-called bounding boxes that also include positional information of objects. Additionally, class labels are assigned to each bounding box which show what has been recognized.

Instance segmentation expands on object detection by not just outputting bounding boxes. It generates binary masks within each bounding box, enabling more accurate identification of the pixels that belong to the object and those that do not. Regions within the bounding box that are not part of the object are disregarded.

Semantic segmentation is a method that outputs a class label for each pixel of the input image. Consequently, the output consists of a mask with the same width and height as the input image. An example is shown in Figure 5d with people and cars being two of the output labels. In this work, semantic segmentation can be used for filtering out the rail track for example.

Panoptic segmentaiton combines semantic segmentation and instance segmentaiton. It makes the difference between so-called "things" and "stuff" [28]. Examples of "things" in Figure 5e are countable objects like cars and people. Examples of "stuff" are the road, buildings or the sky. As with semantic segmentation, this method outputs a pixel-wise classification. This mask also has the same dimensions as the input image. However, while semantic segmentation assigns the same class to different instances of the same objects, panoptic segmentation can differentiate between different "things". Figure 5e shows that each car or person has its unique class label.

Line detection algorithms are usually tailored to filter out lines like road markings or rails. Figure 5f shows a possible output in the road domain of such an algorithm. Even though in the field of line detection a lot of work has been done in the road domain, the main focus of this work is on applications in the rail domain. Furthermore, many state-of-the-art papers include outputs in the form of binary masks with the same dimensions as the input image. While these models solve the line detection problem, they use pixel-wise classification and technically fall into the category of semantic segmentation. In this work, these papers are therefore reviewed in the corresponding section. Solely techniques not based on semantic segmentation are included in the line detection section.

Following a brief description of all the various approaches, it becomes clear that only object detection, semantic segmentation, and line detection are viable options for rail track prediction. Object detection can be used to filter out switches in the image and their states. The information if a switch state is left or right can be useful for predicting the leading rail in an application. Semantic segmentation can be used to identify the track itself on the pixel level. This solution provides a more intuitive solution for train operators where not only a state is outputted, but the track is visualized in the image. Ideally, object detection and track segmentation are combined so that only the train's track is displayed. The third approach, which must not be overlooked are line detection algorithms. While the output and techniques slightly differ from semantic segmentation, they also provide important pixel-wise information about the track. As the other approaches are not suitable or relevant for this use case, they are excluded. The next few sections focus on the three techniques mentioned and describe them in more detail.

5.2 Real-time Object detection

Object Detection models, which result in bounding boxes can be utilized for the detection of rail switches or even switch states. Therefore these models can be of importance for the implementation of the rail track prediction of this work. This section provides a brief introduction to various object detection models. In 2012 [29] showed that deep convolutional networks are capable of extracting abstract feature representations from images in a robust manner. Enabling accurate classification. In 2014 [30] introduced Regions with CNN features Regions with CNN features (RCNN). Since then the development of CNNs can be grouped into two different detection techniques: "two-stage detectors" and "one-stage detectors" [31] [32] [33].

5.2.1 Two-stage detectors

Two-stage detectors usually follow a "coarse-to-fine" process, which firstly includes a region proposal and secondly a classification and a refinement of regions [31] [32] [33] [34]. Well-known examples are the R-CNN [30], Fast R-CNN [35], Faster R-CNN [36], Mask R-CNN [37], and Feature Pyramid Networks (FPN) [38]. Even though these models achieve promising accuracy results, they are highly complex, which increases inference time. Consequently, two-stage detectors are usually unsuitable for real-time-critical applications. Since all of the use cases of the implementation of this work involve real-time capable applications, the inference time is of great importance. Therefore two-stage detectors are not further considered for this work.

5.2.2 One-stage detectors

Single-stage detectors or one-stage-detectors combine the classification and localization in one step making them fast enough for real-time applications. The first single shot detector was the

You Only Look Once (YOLO) [39], which operates with up to 155 FPS. YOLO is the first approach, which reframed the object detection task as a regression problem. The introduced model consists of a single neuronal network, allowing end-to-end training. This model first divides the image into a grid and then simultaneously predicts bounding boxes and the probabilities of classes. Proving to be a fast object detector, [39] presented the beginning of a whole series of real-time capable models. Since, [39] still shows decreased accuracy in the localization of small objects, versions YOLOv3 [40], YOLOv4 [41], YOLO9000 [42], and Single Shot MultiBox Detector (SSD) [43] particularly focused on this issue. However, YOLOv7 [44] and YOLOv9 [45] are among the latest real-time object detection models, emphasizing both high speed and improved parameter utilization [31] [32] [33] [46].

YOLO v7

The YOLOv7 [44] introduced in 2022 is a subsequent work from the YOLOv4 [41]. It surpasses most object detectors in both accuracy and speed, with inferences from 5 FPS to 160 FPS. The main contributions of YOLOv7 [44] are several methods that increase the accuracy without decelerating inference. To achieve that it incorporates a planned re-parameterized strategy, which can be utilized for layers in various models. Furthermore, YOLOv7 [44] also uses new label assignment methods called "coarse-to-fine lead head guided label assignment". Additionally, extend and compound scaling techniques are used. The introduced methods not only increased speed and accuracy, but also decreased the number of parameters of the model by about 40 % [44]. This presents an advantage for this work since the final system is supposed to operate on an embedded device.

YOLO v9

The YOLOv9 [45] is yet another follow-up work from YOLOv7 [44]. Released in February 2024, it is the most recent model in the YOLO series. [45] states that most models lose information through spatial transformations and layer-by-layer feature extraction. Therefore, the YOLO v9 model focuses on reversible functions and information bottlenecks. Consequently, the main contributions of [45] are a Programmable Gradient Information (PGI) concept, which utilizes auxiliary reversible branches, and a Generalized Efficient Layer Aggregation Network (GELAN), which further increases the usage of existing parameters. The proposed models prove to be lightweight while still being accurate and fast, outperforming current real-time object detection models. The characteristics of these models indicate that they are also applicable to this work.

5.3 Semantic Segmentation

The first approach to filter out the rail tracks in front of the train was proposed by <Quelle: 2018: Efficient Rail Area Detection Using Convolutional Neural Network [in TEP]> in 2018. A SegNet <Quelle: SegNet> inspired network for semantic segmentation is extended with mixed pooling <Quelle> and atrous spatial pyramid pooling (ASPP) from DeepLab. After the proposed semantic segmentation network outputs a binary mask with pixel labels being track or no track, a polygon fitting technique is utilized to refine the tracks further.

In 2019 <RailNet: A Segmentation Network for Railroad Detection> introduced the RailNet architecture. This model uses a ResNet-50 <Quelle ResNet> backbone and a fully convolutional network <Quelle in TEP-Net> to segment the rail tracks. The network is designed in a pyramid structure <Quelle: 2017 Feature pyramid networks for object detection>, in which features of every ResNet stage are summed and up-sampled. This combines low and high-level features, resulting in an enhanced performance. <Quelle: RailNet: A Segmentation Network for Railroad Detection> reports higher accuracy than state-of-the-art segmentation models at the time with speeds up to 20 FPS. Tests are made on the introduced RSDS dataset further described in <Dataset section>.

In 2019 RailSem19 <Quelle: RailSem19> introduced the first publicly available dataset for the rail domain. This dataset also includes annotations for semantic segmentation tasks and experimented with a FRRNB model <Quelle: Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes>. This dataset is widely used in the community when working in the rail domain.

<Quelle: 2020 RailNet: An Information Aggregation Network for Rail Track Segmentation> already uses a subset of RailSem19 in 2020. Another RailNet model is proposed that uses a VGG16 backbone and an Information Aggregation Module. <Quelle: 2020 RailNet: An Information Aggregation Network for Rail Track Segmentation> only predicts the rails in RailSem19, other annotations are ignored. The characteristics of rails like placement and structure are considered. Therefore the integrated module is implemented to improve spatial relationship between features on both the vertical and horizontal axes.

<Quelle: 2022 Automated Semantic Segmentation for Autonomous Railway Vehicles> also used RailSem19. The U-Net <Quelle: U-Net> architecture is trained on four different subsets of RailSem19, including 2, 3, 4, or all 19 classes. Additionally, A cropping method is implemented to account for the difference in resolutions between the recommended one for U-Net and one of RailSem19's images.

<Quelle: 2022 Application of Rail Segmentation in the Monitoring of Autonomous Train's Frontal Environment> used the RailSet dataset <Quelle: 2022 RailSet: A Unique Dataset for Railway Anomaly Detection>, which is an extension of RailSem19 for segmentation and anomaly detection tasks. For details of the RailSet dataset please refer to <dataset section>. <Quelle: 2022 Application of Rail Segmentation in the Monitoring of Autonomous Train's Frontal Environment> trained U-Net <Quelle: U-Net> and FRNN <Quelle: FRNN> and incorporated

horizontal flips and zooming into the data augmentation.

In 2020 <Quelle: Railroad semantic segmentation on high-resolution images> proposed a U-Net-inspired <Quelle: U-Net> semantic segmentation network. It combines a ResNet-34 backbone and includes connections to the upsampling blocks. At the deepest level, a spatial pyramid pooling (SPP) <Quelle: Spatial pyramid pooling in deep convolutional networks for visual recognition> and on the skip connections squeeze-and-excitation blocks <Quelle: wie bei mobilenet> are used. Tests on RailSem19 <Quelle: RailSem19>, show that it outperforms <Quelle: 2019 RailNet: A Segmentation Network for Railroad Detection> in accuracy with a speed of 20 FPS. Additionally, <Quelle: Railroad semantic segmentation on high-resolution images> introduced the concept of "possible tracks", which are all paths the train could follow under the assumption that the state of switches cannot be determined. For this, a rule-based post-processing algorithm is proposed. <Figure xy> shows the concept of possible paths.

<vielleicht figure possible tracks von Quelle: Railroad semantic segmentation on high-resolution images>

In 2023 <Quelle: TPE-Net: Track Point Extraction and Association Network for Rail Path Proposal Generation> further investigated the task of possible tracks and introduced Track Point Extraction and Association Network (TPE-Net). It consists of a DenseNet-inspired architecture, which is trained and tested on RailSem19 and outputs regressed heatmaps besides the segmented rails. An example of such a heatmap is an image with one channel that shows the probability of each pixel being within a rail track. The segmentation mask and the heatmaps are then used by a complex post-processing approach. This process includes track point clustering, the creation of track segments, and the creation of a path tree, which is then used to generate all possible tracks. Refinement is done by removing redundant paths and polynomial fitting. Because of the complexity of this system, <Quelle: TPE-Net: Track Point Extraction and Association Network for Rail Path Proposal Generation> reports speeds up to 12 FPS making this system unsuitable for real-time applications. Additionally, it is stated that problems arise when switches are present.

In 2022 <Quelle: RailVID: A Dataset for Rail Environment Semantic> proposed the RailVID dataset, which consists of infrared images instead of RGB data to improve the system's abilities in challenging situations such as the absence of ambient light. The dataset is described in <Dataset section> in more detail. After collecting data, <Quelle: RailVID: A Dataset for Rail Environment Semantic> experimented with widely used CNNs for semantic segmentations, like CGNet <Quelle>, DeepLabv3+ <Quelle>, and BiSeNet <Quelle>. Additionally, an improved BiSeNet architecture is proposed with consideration of the infrared data. Performance is enhanced by added layers to fuse low-level features.

In 2021 <Accurate and Lightweight RailNet for Real-Time Rail Line Detection> proposed another architecture called RailNet. It consists of an Encoder-Decoder structure incorporating

depth-wise convolutions and a Segmentation Soul block, inspired by the context embedding block of BiSeNetV2 <Quelle: BiSeNetV2>. Additionally, a port processing algorithm is utilized that is based on sliding window detection. <Accurate and Lightweight RailNet for Real-Time Rail Line Detection> reports speeds up to 74 FPS proving that this system is real-time capable.

A topic that could be interesting is anomaly detection because many state-of-the-art approaches use semantic segmentation as a preprocessing step. <2021 Near Real-time Situation Awareness and Anomaly Detection for Complex Railway Environment> utilizes a BiSeNet architecture for segmenting rails, which is tailored for the anomaly detection task. This network can detect small objects on the track. Therefore accuracy is preferred over speed, resulting in a system that is not real-time capable.

5.3.1 Example Überschrift

blindtext

5.3.2 Example Überschrift

blindtext hallo

- Image Classification
- Object Detection
- Semantic Segmentation (detection und keine direction prediction) hat normalerweise probleme wenn mehrere rails in der scene sind –> weichen schwer trennbar weil der pixelhaufen dazwischen irgendwas ist
- Instance Segmentation
- panoptic Segmentation
- line detection (detection und keine direction prediction) anzahl der rails in einer scene ist das problem

Für die nächstem kapitel:
was gibt es?
was gibt es in der rail domaine?

5.3.3 Rail Segmentation oder Semantic Segmentation (2 bis 3 Seiten)

- DeepLab V3+
- AdapNet++
- ...

5.3.4 Line detection algorithms (2 bis 3 Seiten)

Rail Detection: An Efficient Row-based Network and A New Benchmark –> hat auch Lane Detection & Railroad Detection

5.3.5 Backbone architectures 2 Seiten

- EfficientNet 1/2 Seite
- ResNet 1/2 Seite
- MobileNet 1/2 Seite
- DenseNet 1/2 Seite

5.3.6 mögliches Baseline Paper vielleicht (wenn die Aufteilung geändert werden muss)

5.3.7 verwendetes Baseline Paper vielleicht (wenn die Aufteilung geändert werden muss)

5.3.8 Temporal Models (2 Seiten)

- LSTM -> 1 Seite
- GRU -> 1 Seite
- ...

5.4 Network architectures

One of the first CNN architectures was the **Le-Net5** [47]. This CNN was proposed to classify handwritten digits. It used 5 convolution and pooling layers. 14 years later CNNs were re-discovered with the introduction of **AlexNet** [29] in 2012. In the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [48] in 2012 **AlexNet** outperformed traditional algorithms and showed the potential of CNNs. Consequently, it is often considered the starting point of the CNN era in the literature. Since then research has shown a rapid development of CNN architectures, consistently striving for greater accuracy or speed [49].

Several major advancements up to now include:

- In 2014 two popular architectures were introduced, which still are often used for state-of-the-art comparisons. Firstly, the **Visual Geometry Group Network (VGGNet)** [50] used small three-by-three filters and many layers providing a deep network architecture. Secondly, the **Inception** network or **GoogLeNet** [51]. To extract features in a range of various scales, this architecture utilized different filter sizes.
- In 2015 the **ResNet** architecture [52] introduced the concept of residual learning with shortcut connections. By skipping layers, this framework enabled the training of deeper networks with over a hundred layers [49].
- In 2016 **DenseNets** [53] was introduced to further deal with the problem of vanishing gradients. This architecture implemented a connection between each layer and every following layer ensuring maximum information flow [49].
- In 2017 **MobileNets** [54] [55] [56] was developed for applications in mobile devices. Since hardware with limited capabilities is used, the main focus of MobileNet architectures lies on efficiency. The utilization of depthwise separable convolutions allowed small model sizes and fast latencies [49].
- In 2019 **EfficientNets** [57] was introduced to further explore greater trade-offs between accuracy and efficiency, by using scaling methods for model architectures [49].

Since the CNNs are a part of a rapidly developing research field, more recent architectures are more interesting for this work. Because they already solved issues that exist in older models. Therefore, only the last four mentioned CNNs are described in more detail in the next sections.

5.4.1 ResNets

A widely used model architecture in the community is ResNet [52]. Up to this architecture CNNs suffered from the vanishing gradient problem, particularly when the model consists of a deep network. A neural network is typically considered deep when it comprises a large number of consecutive layers. To solve that issue ResNet introduced the residual block that is also visualized in Figure 6. This block enabled effective training of models with up to 152 layers. Compared to VGGNet ResNet is 8 times deeper but still less complex. ResNet showed a great performance gain and was therefore granted first place in the classification task of the ILSVRC 2015. ResNet is supported in Pytorch, which includes variants with 18, 34, 50, 101, and 152 layers [58]. However, ResNet models are still resource-intensive because of their size [49].

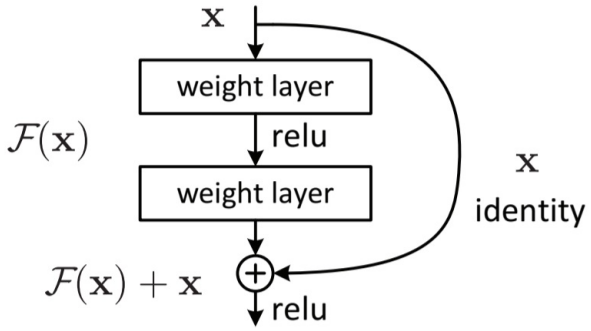


Figure 6: ResNet residual block [52]

Figure 6 schematically represents the ResNet's skip connection. The block has two processes. One consists of two layers and the other one performs identity mapping. After that, the outputs are added together.

5.4.2 DenseNets

DenseNet [53] further explores skip connections and implemented architecture blocks in which each layer is connected to every following layer. An example of a so-called "dense block" is shown in Figure 7. Compared to ResNet, this model connects outputs via concatenation instead of addition. Between dense bocks, there are the so-called "transition layers", consisting of one convolutional and then one pooling layer. This architecture allows deeper CNNs which result in more accuracy and efficiency. DenseNet showed great results with the introduced technique because their blocks further tackle the vanishing gradient problem with feature reuse. PyTorch includes DenseNet model variants with 121, 161, 169, 201 [59].

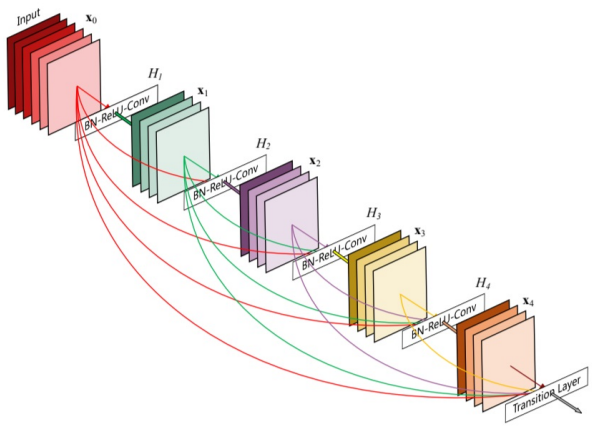


Figure 7: DenseNet dense block [53]

5.4.3 MobileNetV3

The goal of the MobileNet series [56] is to efficiently operate on limited hardware for example on mobile devices. Therefore the main focus of these models is to be as lightweight as possible [49], meaning the reduction of parameters and ensuring real-time capabilities while maintaining comparable accuracy. There are 3 versions of MobileNets and each one builds upon and extends its predecessor. To achieve a reduced size version 1 utilizes depthwise separable convolutions. These blocks of convolutions consist of two steps. First, a depthwise convolution, where a kernel is deployed on each channel, followed by a pointwise convolution, where a 1x1 kernel combines the output from the first step [49]. MobileNet V2 then added a layer with a 1x1 kernel before the depthwise convolution and a residual connection to create its so-called bottleneck blocks. A bottleneck block from MobileNetV2 is shown in Figure 8a.

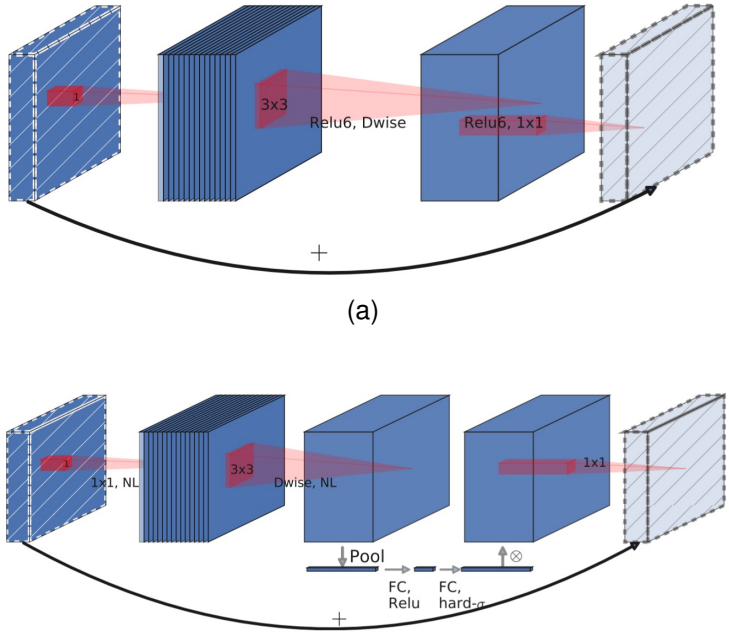


Figure 8: MobileNet V2 and V3 Blocks [56]: **(a)** MobileNetV2 bottleneck block, **(b)** MobileNetV3 bottleneck block with added Squeeze-and-Excite block

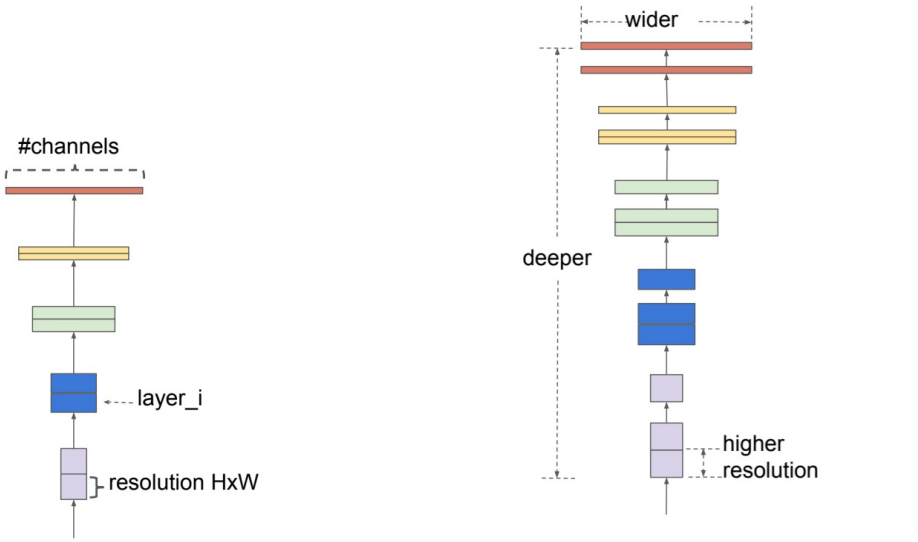
MobileNet V3 added optional blocks called Squeeze-and-Excite, which is illustrated in Figure 8b. These blocks can be added to any CNN and consist of a global pooling layer and two fully connected layers with a ReLu and a Sigmoid activation function respectively. After that, the output of this additional block is multiplied with the input feature map of the SE block. This enables the weighing of specific channels [60]. Additionally, for the MobileNet V3 a Network Search is used to find optimal network structures and experiments with various activation functions have been conducted [56].

MobileNets allow flexible usage with the so-called width or in V3 depth multiplier, which is a hyperparameter for controlling the number of feature maps in layers. MobileNet V1 and V2 additionally offer the resolution multiplier, which controls the resolution of layers. Both the MobileNetV3_small and MobileNetV3_large are supported in PyTorch [61].

5.4.4 EfficientNets

EfficientNet [57] further investigates methods to find CNN architectures optimized for efficiency. Inspired by MobileNet V3 the models also deploy residual bottleneck convolutions. [57] calls such a block an "MBConv". Since [57] observed the correlation between performance improvement and model scaling, the idea is to strategically adjust the depth, width, and resolution of a network. Figure 9b visualizes these parameters in an example network. Consequently, a scaling method is introduced, which uses a so-called "compound coefficient". The resolution as well as the depth and width are uniformly scaled with this technique, resulting in eight variants

of EfficientNet. The different versions, starting with B0 and ending with B7, have increasing parameters. Since this work focuses on lightweight architectures the most interesting models are the first four (B0, B1, B2, B3). All variants of the EfficientNet are included in the PyTorch library [62].



(a) network before scaling method

(b) scaling parameters of a network acording to [57]

Figure 9: Systematic visualisation of scaling parameters of a network [57]

5.5 Datasets

Datasets are essential for the development of autonomous driving systems, particularly for training and testing algorithms or neural networks. Typically, raw sensor data is collected from real-world driving scenarios, providing a realistic environment that reflects potential situations the system may encounter in future applications. This allows for more accurate modeling and evaluation of the system's performance under real conditions. A common approach to solving problems in autonomous driving systems is using vision-based machine learning algorithms [26, S. 1221]. These applications typically rely on camera-based data to address various challenges [26, S. 1221].

Since, autonomous vehicles are increasingly considered a groundbreaking technology of the future [23] a lot of work is put in the development of such systems. However often the main focus of this quickly evolving field is on road vehicles, like cars or trucks. Therefore, most publicly available datasets focus on this application and primarily reflect scenarios in road traffic [26, S. 1221].

Classification Datasets

There are a couple of public datasets for different Computer Vision (CV) applications. The data of the following datasets are especially gathered for classification tasks.

Table 3: Datasets for Classification

Dataset	Labels	Number of relevant images
CIFAR 100	trains	600
PASCAL VOC2012	trains	544
Microsoft COCO	trains	3745
1000 ImageNet	electric_locomotive, steam_locomotive, bullet_train	6722
Open Images Dataset V4	train	9284

CIFAR 100 dataset [63] consists of small images with pixel size 32x32. This dataset includes 600 images with the label *trains*. In *PASCAL VOC2012* [64] there are 544 images labeled *trains*. *Microsoft COCO* [65] has 3745 images with the class *trains*. Additionally, there are classes like *traffic lights* and *stop sign*, however theses are not useful to the task of this work because they are from the street domain and not the rail domain. *1000 ImageNet* [66] includes labels like *electric_locomotive* (4330 images), *steam_locomotive* (1187 images) and *bullet_train* (1205 images). *Open Images Dataset V4* [67] consists of more than 9.2 million images, annotated with bounding boxes. Included are 10506 *train*-labels in 9284 images. However, there are no

other significant labels relevant to the rail domain. Additionally, the dataset contains labels for toy trains, which could pose potential challenges.

Semantic Segmentation

Semantic segmentation labels are often refereed to as dense or pixel-wise annotated data. These datasets are characterized by the fact that each pixel in their images is assigned to a class.

Table 4: Datasets for Semantic Segmentation

Dataset	Labels	Number of relevant images
Cityscapes	rail track, train	284
Mapillary Vistas	construction-flat-rail-track, object-vehicle-on-rails	710
COCO-Stuff	platform, railroads, train	8615
KITTI	rail tracks, train	65

The *Cityscapes* dataset [68] is commonly used for benchmarks when it comes to road scenes. It has 35 different labels of which two are *rail track* (131 labels in 117 images) and *train* (194 labels in 167 images). The *rail track* does not differ between the rails and track bed. *Mapillary Vistas* [69] also has more labels, but again only two are rail related ones. *construction-flat-rail-track* is annotated in 710 images and *object-vehicle-on-rails* occurs 272 times. *COCO-Stuff dataset* [70] includes the same 182 classes like the *Microsoft COCO* [65] but as dense labels. The rail relevant ones are *railroad* (2839) and *train* (4761). There is a third rail related label *platform*, however this is a very general label because this can by any plane. *KITTI* [71] has the same dense labels like *Cityscapes* [68]. Likewise, the rail relevant ones are 47 *rail track* and 18 *train* labels.

These are commonly used datasets in CV tasks. However there are there are three main issues when it comes to solving the track prediction use case presented in this work. Firstly, there is not enough data because the amount of included rail relevant labels is relatively low in each dataset. Secondly, the labels present are not suitable for training a track prediction algorithm. In this case only the rails, rail tracks or track beds are needed. Thirdly, the images of the presented datasets are taken out of passengers and pedestrian views. Additionally there are some road views [72].

The datasets mentioned before are very general with a vast amount of different labels. However there are some datasets specially captured for the rail domain. As with the other datasets, it is important to consider what specific tasks the datasets are intended to be used for. There are some datasets captured in the birds eye view to detect damages like cracks in rails [73] [74] or even some to detect garbage in grooved rails [75]. Then there are datasets in ego-

perspective of the train driver like *FRSign* [76] or *GERALD* [77], which are created for detecting different traffic lights on French and German railways.

This work deals with Rail Track Prediction, so the system should predict the direction of the track in front of the train. For this particular use case it is most advantages when the dataset is recorded out of the driver cabin in ego-perspective, because it offers a clear sight of the rails in front of the train. Therefore the data has to reflect scenarios, which are comparable. Since the view of the captured images represents a key factor, the before mentioned dataset become unsuitable. An additional reason why these datasets cannot be used for this work is the fact that they are created for different use cases. Other datasets which deal with the perception in a rail domain environment and are captured in the right perspective are discussed in the following paragraphs.

RailSem19

RailSem19 [26] is the first publicly available dataset fitted for environments in the rail domain. It consists of 8500 annotated images which are gathered from YouTube videos. All of these images are captured in the ego perspective of the train driver, which makes it suitable for the use case of this work. Additionally there are both bounding box labels and dense labels included for object detection and semantic segmentation. The bounding box labels are: *guard-rail; rail; traffic-signal-front; traffic-signal-back; traffic-sign-front; crossing; train; platform; buffer-stop; switch-indicator; switch-static; switch-left; switch-right; switch-unknown*. Important for predicting the direction of the train are the switch labels, because it gives valuable information. The *switch-unknown* label is used when there is a switch visible but it is unclear in which direction the train would proceed. The presents of this label is mainly due to the high noise levels of the images of YouTube videos. The dense labels of RailSem19 are: *road; sidewalk; construction; tram-track; fence; pole; traffic-light; traffic-sign; vegetation; terrain; sky; human; rail-track; car; truck; track-bed; on-rails; rail-raised; rail-embedded; void*. In the case of dense labels the labels *tram-track; rail-track; track-bed; on-rails; rail-raised; rail-embedded* are of importance for predicting the direction of trains and trams. Another advantage is that very diverse environments have been used for this dataset. The creators of *RailSem19* took images from 38 different countries in all four seasons and weather conditions. Additionally, the focus was not only on rails but also on trams, providing a very diverse reflection of rail scenarios and not limiting the use on a specific use case.

Figure 10: RailSem19 dataset examples. First row raw images. Second row dense GT [26].

RailVID

Another dataset that focuses on the detection of rails is the *RailVID* dataset [78]. The goal of this project is to detect rail tracks and obstacles on the rails, which can lead to possible hazardous situations. With a functioning system, fully automatic train operation is aimed for. The *RailVID* dataset is a collection of 1071 images with the following labels: *background, railway, car, people*. Since, the area of application is on the "Suzhou Rail Transit Line 1" in Jiangsu Province, China all data is captured there. *RailVID* is a collection of infrared data captured with the AT615X infrared thermal instrument from InfiRay. The decision to use infrared data and not RGB images is because it is more robust against challenging imaging conditions, like darkness at night, fog, rain and direct light disturbance. Since, this dataset only consists of infrared data and in this work Red Green Blue (RGB) data should be used, *RailVID* cannot be used fro training. An additional issue is, that the dataset is recorded only on a specific Chinese line. This is advantageous for this particular use case, but it could become an issue if the system were to be deployed elsewhere.

Figure 11: Example images and GT of RailVID dataset [78]

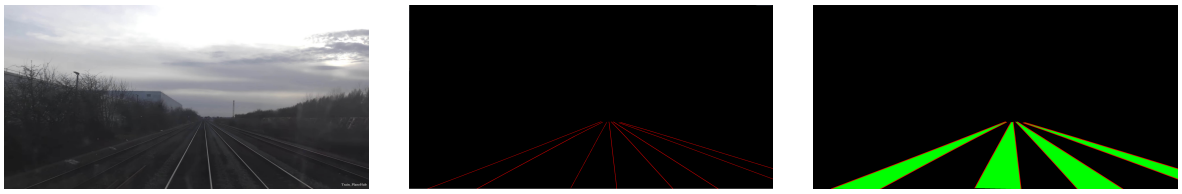
RailSet

[79] and [80] presented the *RailSet* dataset, which is divided into two sub sets: RailSet-Seg for segmentation and RailSet-Ano for anomaly detection [79]. Both of them are captured in the ego

perspective of the train driver [79] [80]. The idea is to firstly detect the railways using semantic segmentation and secondly using positional information of the prediction for the creation of the anomaly dataset [79]. RailSet-Ano is a collection of 1100 images of railway defects like rail discontinuity and holes in the rail bed. Some anomalies are taken from a other images and pasted on images of RailSet-Seg and RailSem19 dataset others are generated with a network [79]. Since, RailSet-Ano deals with a different use case than the one in this work, this particular data cannot be used.

On the other hand, RailSet-Seg fits the problem. It consists of 6600 images of normal situations. The images are collected from 23 YouTube videos with a collective duration of 15 hours. It includes two labels: *rail* and *rail-track*. Besides the use of RailSet-Seg for the creation of RailSet-Ano, an additional motivation is to include more complex scenes of the rail domain than RailSem19. That is why the focus of RailSet lies on scenarios with poor weather conditions or lighting conditions. Furthermore, images are included in which the rails are not visible at all, like in tunnels without lighting or in snowy scenes. Additionally, it was ensured that the videos were recorded by different cameras and from different mounting positions [79] [80].

An advantage of this dataset is that it can be joined with RailSem19, when combining four specific labels. *trackbed* and *rail-track* from RailSem19 have to be transformed into RailSet's *rail-track* label and *rail-raised* and *rail-embedded* become the *rail* label. This method leads to more data for the training and validation which could be an advantage. However, it also shows the disadvantage of this dataset for the specific use case of this work. RailSet exclusively addresses railway and not tram scenes [80]. Since, the goal is to target a broad applicability, leaning towards tram scenes this dataset is not used for this work.



(a) (b) (c)

Figure 12: RailSet-Seg example with annotations [79] [80]: **(a)** raw-image, **(b)** rail class, **(c)** rail and rail-track class

RSDS

The creators of the Railroad Segmentation Dataset (RSDS) [27], tried to solve the railroad detection problem with a segmentation approach. Because there was no publicly available dataset for this task at the time, they had to construct their own. RSDS is captured from the ego perspective of the train driver and is a collection of 3000 images. They used 2500 for training, 200 for validation and 300 for testing. The dataset only includes one *railroad* label. It is described that the labels only incorporate pixels between the two rails and intentionally ignore railway sleepers outside this area. Images are of size 1920 x 1080. Although it is not

mentioned in the paper, it seems as if all images of RSDS are captured from a specific rail line in China. Additionally, from the images in the paper alone one can tell that this particular rail line has very distinctive structural characteristics. The colors are bright and it seems like the area between the rails is mostly concrete, which is unusual for most railways. 13 shows that structures besides the rails are specific too. One additional detail of this dataset, which can present a disadvantage for this work, is that switches are not addressed. Since this dataset only covers very specific railroads with unique characteristics and additionally does not address switches or other situations where the track splits, RSDS is not further considered for this work.

(a) (b)

Figure 13: RSDS example with annotation [27]: **(a)** raw-image, **(b)** ground truth

Rail-DB

A very similar dataset is Rail-DB [81]. This dataset is a collection of 7432 images, which are taken from 15 videos. The labels in this dataset are consist of poly lines, which represent all existing rails in an image. Additionally the the poly lines all have different classes. This way there is not only one rail class, but as many classes as there are rails in one image. The labeling policy specifies that the central rails are marked with the annotations 1 and 2. Additional rails are labeled with rising numbers. Since this dataset includes poly lines and not binary masks, this dataset can be used for training line detection algorithms. Compared to RSDS, this dataset includes not only lines and curves, but also rail switches. Additionally, the images are taken in various conditions and scenes. 14 shows example images of Rail-DB's different scenes. However, it seems that the images again have very specific characteristics like in RSDS. 14 also shows, that all images are captured on a Chinese line. Even though [81] presents a very interesting approach for solving the rail detection problem, because of the before mentioned specific characteristics it is not used for this work. Even the author of [81] states in the GitHub repository [82], that this project fails to generalize on example videos, where the scene looks different from the one the dataset is captured on.

Figure 14: Rail-DB [81] images with annotations in different conditions

OSDaR23

Another dataset it the OSDaR23 [83]. This dataset is a collection of 21 sequences, which are split into 45 subsequences. Several sequences are short ones with 10 frames each, some are longer with 40 to 100 frames. In total there are 1534 labeled scenes in this dataset. Since, OSDaR23 is captured with 9 cameras (RGB and Infrared (IR)) in different angles, one lidar and one radar sensor, the total number of frames are 1534 * 11 = 16874. Additionally, position and acceleration sensors are used. Because also lidar and radar is used, this dataset offers 3D data as well. OSDaR23 consists of 20 different labels. These labels include the rail context, like *track*, *switch* or *train* and the environment, like *person*, *animal*, *bicycle*, *smoke*, *flame* or *crowd*. The annotations for the environment can be used for safety applications. For more details please refer to *TABLE V* in [83]. 15 shows the 11 different frames of each sensor and 16 shows an example of an annotated scene. As illustrated in 16 the *track* label and the *switch* label only offers positional information about their presents, but do not include any information on the direction of the train. Furthermore, there is no information that distinguishes the train rails from the adjacent rails. Therefore OSDaR23 can only be used for the detection of rails, but not the track prediction. Moreover, the capturing of this dataset only took place on rail roads in Hamburg, Germany. No tram scenes are included. Due to the necessity of re-labeling for this work and the fact that only data from Hamburg is offered, OSDaR23 is not used.

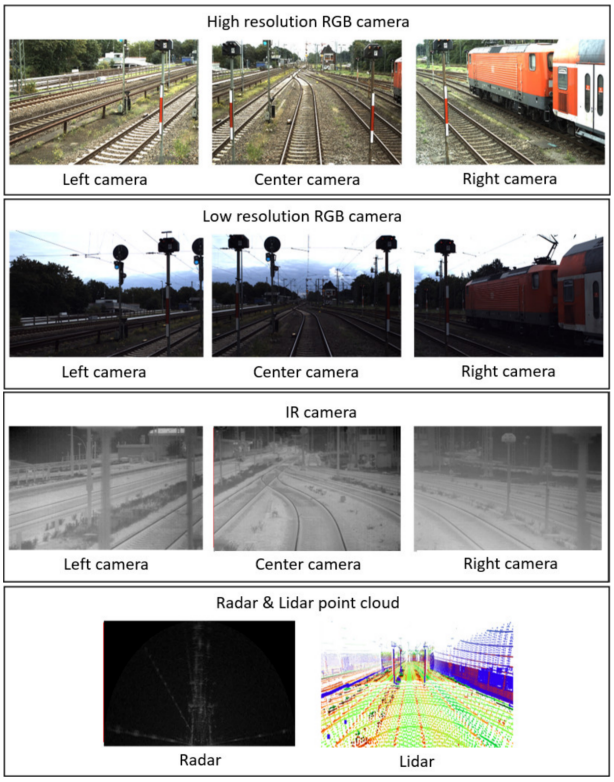
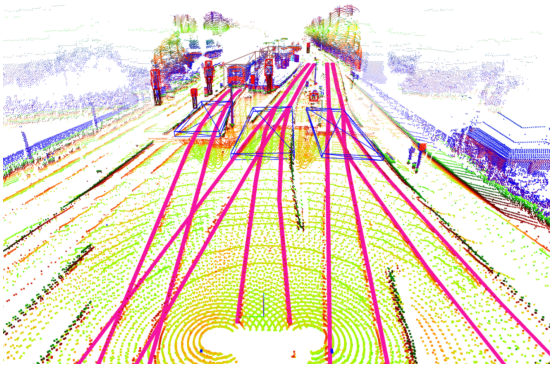
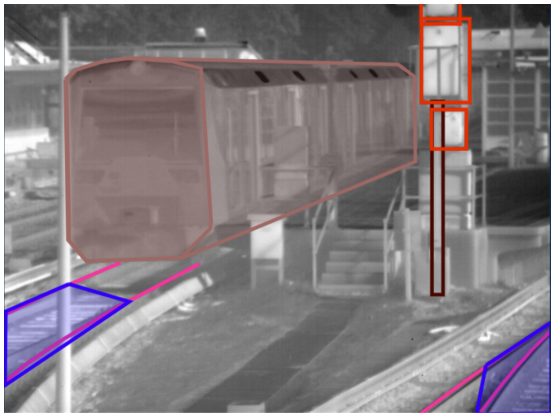


Figure 15: OSDaR23 data from all different sensors [83]

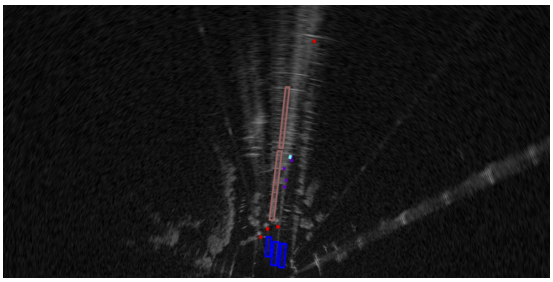
(a) RGB center camera



(b) merged lidar point cloud



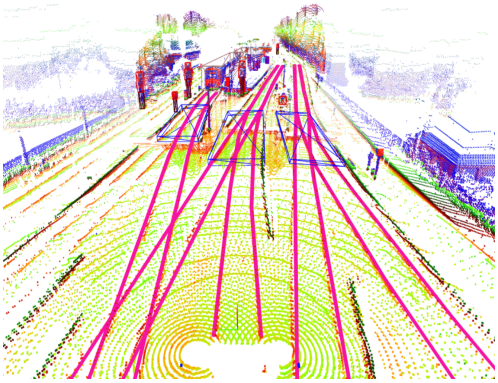
(c) IR center camera



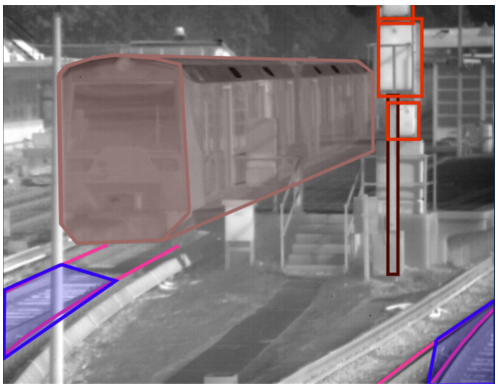
(d) Radar (zoomed)

Figure 16: OSDaR23 annotated scene [83]

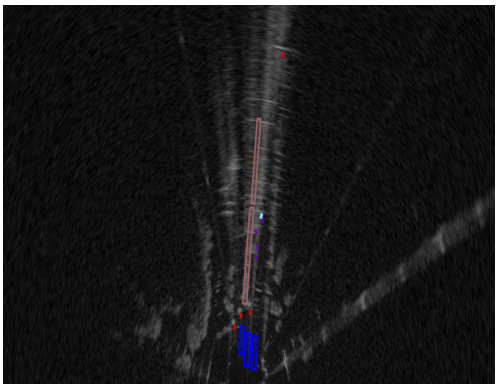
(a) Bild 1



(b) Bild 2



(c) Bild 3



(d) Bild 4

Figure 17: OSDaR23 labels skaliert

TEP-Net dataset

[1] presents the Train Ego Path (TEP)-Net dataset. The problem this paper aims to solve is rail track prediction. This differs from all previously mentioned datasets. No dataset but [1] provides information, which distinguishes possible rails in the image from the one the train actually follows. Since RailSem19 is the most popular dataset in the rail domain, it is used as initial point. A total of 7917 images were taken from RailSem19. The remaining 583 were excluded because they are taken from unusual perspectives or it is unclear which track the train is on or would continue on. Examples of such images are shown in 18. These images are excluded simply by not annotating them. For annotation a new labeling format is created. Two classes *rail_right* and *rail_left* give information about where the tracks of the train run. All other rails which might be in the image are ignored. These two classes consist of poly lines, which are annotated by the corresponding x and y pixel coordinates of the image. Only the tracks on which the train is located are labeled. Even if a switch appears in the image and the train would travel over it, only the tracks in the correct direction are further labeled. This way, switches are indirectly included in this dataset, providing information on the direction a train would continue, even though there is no explicit label for switches. The poly lines start from lowest pixel row

of the images and extend up to a specific horizon line. Above this horizon further labeling is not possible. This may occur for various reasons. The first reason can be an obstruction of the view by the environment or other trains, as shown in 19. Since the images are sourced from YouTube videos, it is also possible that the resolution becomes too low in the distance for separately identifying both rails. Another reason is when it is not possible to determine the direction in which the track continues based on a switch present in an image. This can happen due to low resolution or unfortunate camera angles. These cases may be labeled as *switch-unknown* in the original RailSem19 dataset for example. Here, the polylines stop before the switch. The polylines from this annotation can then be converted into several different labels using preprocessing algorithms. On one hand, they can be directly used as polylines. On the other hand, they can be transformed into a mask for segmentation tasks by filling in the area between the lines. A third application would be to convert this mask into a grid for classification tasks.

Figure 18: Examples images from RailSem19, which are not included in the TEP-Net dataset due to unclear circumstances about the trains direction. [1]

Figure 19: TEP-Net dataset example images with annotation [1]

notes wo alles ist

- Railroad Segmentation Dataset (RSDS) –> in RailNet: A Segmentation Network for Railroad Detection fertig
- RailSem19 in RailSem19 fertig

- RailVID in RailVID fertig
- RailSet in RailSet & Application of Rail Segmentation in the Monitoring of Autonomous Train's Frontal Environment fertig
- Rail-DB (compared to RSDS) –> Rail Detection: An Efficient Row-based Network and A New Benchmark fertig
- OSDaR23 in OSDaR23: Open Sensor Data for Rail 2023 fertig
- TEP-net dataset fertig
-
- RailSet -> Segmentation & Anomaly detection
- Application of Rail Segmentation in the Monitoring of Autonomous Train's Frontal Environment -> RailSegmentation (only rails and trackbed)

5.6 Baseline Paper

Due to the unique approach of [1], it does not fit in any of the sections above. To do justice to this unique contribution, it is addressed in a separate section highlighting the paper's specific insights and relevance.

To the best of the author's knowledge, [1] is the only paper in the literature, which makes a distinction between rail detection and train track predictionIn this paper, the train's rails are defined as the "ego-path" and others are ignored. To realize a system with this output [1] presents a novel regression-based approach, inspired by autonomous driving applications for road cars.

The main contributions of this work consist of several aspects. First, unique annotations are created specifically for this project. Furthermore, a data augmentation strategy is presented, incorporating two distinct cropping techniques for training and inference. Additionally, a new Regression-based model architecture is developed, to meet the requirements of the novel problem formulation. A custom loss function is introduced.

In the next few sections, each of these novelties is described in more detail.

5.6.1 Annotations

A new dataset is required for this. [1] works with its annotations and the RailSem19 dataset. This dataset is in the so-called "ego-view", so the cameras are positioned in the driver's cabin and view the rails in front of the train. For further details of the dataset utilized in this project, refer to section 5.5.

5.6.2 Data augmentation

This section describes both the data augmentation strategy for training and the cropping augmentation for inference.

Data augmentation for Training

Three data augmentation techniques are implemented to train the model. The first two are commonly used in CV tasks, including random horizontal flips and traditional image adjustments like brightness, contrast, saturation, and hue variations. For this, the standard ColorJitter method from PyTorch is used [84]. The third augmentation is the cropping mechanism, which reduces the image to the most relevant part. Randomness is also introduced to enhance generalization.

Figure 20 visualizes the cropping algorithm steps schematically. At first, there are two green lines, which follow the train's ego-path and represent the GT in the form of polylines. Then the yellow rectangle is computed with this information, being the smallest possible rectangle around the GT. In the next step, the orange box, the start pixels of the rails GT are centered by expanding the yellow borders in one direction. Important to mention is that in this tailored

dataset the rails always begin in the first row at the bottom of each image. After that margins are added shown by the red rectangle. These margins are predefined with hyperparameters. After calculating the borders of the outermost box variability is added to enhance generalization and prevent biases. The left, right, and top borders are moved by a distance calculated with a normal distribution, which is defined by hyperparameters. This introduced variability follows the policy, which ensures that the starting pixels of the rails at the bottom stay in the cropped image. Figure 20 illustrates four augmented variations of the top image in the bottom row.

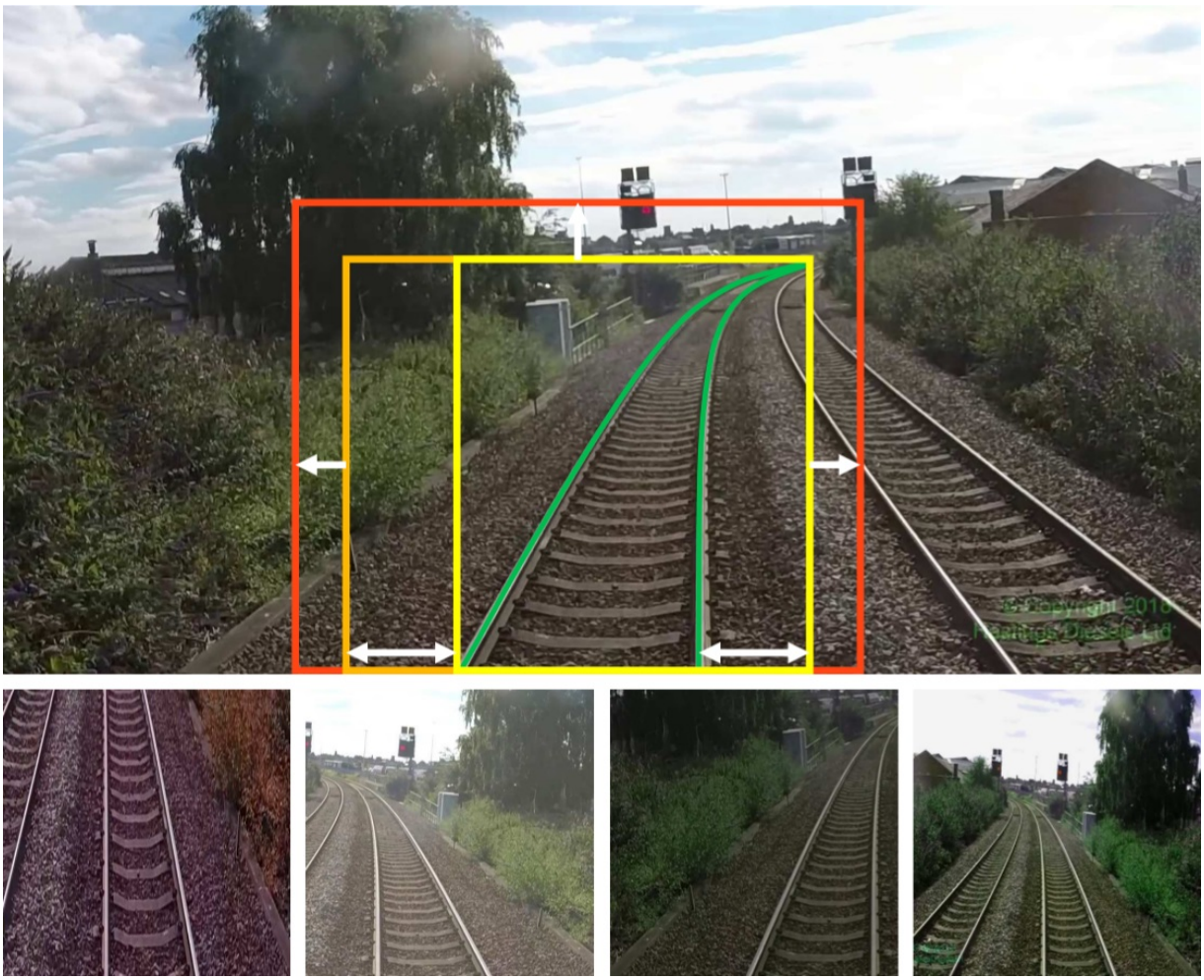


Figure 20: Data augmentation of TEP-Net [1], including the image horizontal flips, the color variations and the cropping mechanism.

Cropping in inference

In [1] only cropped images are used for training the model to minimize the images to their most relevant Region of Interest (ROI). For training and validating the model on the dataset, the GT allows the computation of crop coordinates (left, top, right). However, GT data is unavailable in practical applications like video inference. As a result, the crop coordinates must be, either predefined by a user or computed on the fly.

In a real-world application the ROI in the images changes, even when the camera is mounted in a fixed position. Because of these perspective shifts when the train drives right or left turns, it easily can become unsustainable to have predetermined crop coordinates. Figure 21 visualizes these perspective shifts. The tighter the curve, the worse the shift. The approach of [1] to solve this particular problem is the so-called "Autocrop". This developed technique starts with the whole image, so the initial crop cords are the image borders. After the initialization, the three coordinates (left, top, right) are updated according to the prediction of every new frame. In more detail, new coordinates are calculated with a running average of the smallest rectangle around the prediction plus the predefined margins from the training.

Figure 21: Example of perspective shifts with three different scenarios: left curve, right curve and straight rail [1].

5.6.3 TEP-Net Model

In the literature, the particular problem of rail detection is usually solved with segmentation-based approaches. One limitation of these approaches is the lack of distinction between all rails visible in the image and the rail the train continues. The unique approach introduced by [1] takes a regression-based approach, which restricts the model to predict a single track. The idea comes from various lane detection methods in autonomous driving applications for road cars and is fitted the rail domain.

Although rails can be represented by second or third-degree polynomials in curves, they may also take more complex geometric shapes. Hence, limiting the output to presumed forms is discouraged. Therefore, the method used in [1] employs spline interpolation to describe such complex structures.

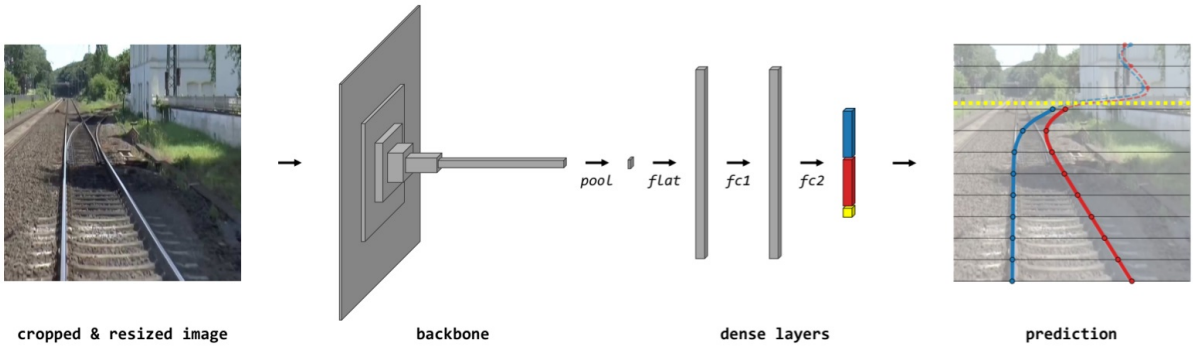


Figure 22: TEP-Net model architecture[1]. The input of the model is a cropped and resized image and the output of the model are the x-values for the left and right rail on each anchor line plus an additional value for the y-limit.

As shown in the prediction of Figure 22, a set of horizontal lines are overlayed in the cropped image. The number of y-lines or "anchors" is determined by a hyperparameter. They are uniformly distributed along the y-axis. For each line, two x-values are predicted. One for the left rail and one for the right rail. The second and fourth images in the bottom row of Figure 20 and the prediction image of Figure 22 show that rails do not necessarily cross with anchor lines at the top of image crops. Therefore, an additional y-limit is predicted, which gives information up to which anchor the rail should be detected. Anchors and their x-values above this horizon line do not hold valuable information and are ignored.

For this novel regression task, a new model architecture is created. For this model widely used backbone architectures like ResNet and EfficientNet are chosen. These backbones extract relevant features from the cropped images reducing the spatial dimensions and increasing channel size to a high number. After that, the feature map's number of channels is reduced to a predefined size with a 1x1 Conv2d layer [85] and flattened to a vector. This feature vector serves as the input for the prediction head, which consists of two fully connected layers [86] in series. The size of the linear layers is set with a hyperparameter. In the last layer, a reduction leads to the resulting prediction vector with the dimension $2 \times H + 1$. H is the number of anchors. This vector includes the entire information of one prediction. The first set of values with size H is for the left rail, another H set for the right rail and the $+1$ is for the last values being the horizon line.

This architecture's policy for value ranges does not restrict the x-values in any way. This way predictions can be outside of a crop and the model can learn that sometimes the rails extend out of the viewed field. The y-limit on the other hand is constrained to a range between 0 and 1 with a Sigmoid function.

The introduced model architecture can be classified as an end-to-end framework. This means the model can be trained and used for inference without any steps in between. It takes in raw data and results in a complete prediction.

5.6.4 Loss function

One of the main contributions of [1] is the loss function, which is tailored to the particular problem formulation of the regression-based approach. The final loss function Equation 3 [1] consists of two sub-functions, where the outputs are added together, with the y-limit loss weighted by a multiplication factor of $\lambda = 0.5$ before summing.

$$L = L_{traj} + \lambda \times L_{ylim} \tag{3}$$

The two sub-functions are the trajectory loss L_{traj} and y-limit loss L_{ylim}. For the following functions, $\hat{\mathbf{X}}$ and $\mathbf{X}$ represent the predicted and GT x-values. These include the points for the left and right rails. For the y-limit, the predicted and GT values are $\hat{y}_{lim}$ and y_{lim}.

Trajectory Loss L_{traj} determines the error between the predicted and actual x-values. To calculate this error, the sum of smoothL1 is used as shown in the numerator Equation 4. This is done with a $\beta_1 = 0.005$. As shown in figure Figure 23, this function allows a linearly proportional relationship between loss and error. Additionally, for noise in the data, it includes a smooth transition at zero.

One issue of the front view perspective is the so-called linear perspective effect. This effect occurs when rails continue into the distance. The same pixel error represents a small distance when near the camera's capturing point and a larger distance when far away. This ratio grows linearly the greater the distance to the camera. To consider this effect, the results of the $L_{rails}(\hat{\mathbf{X}}_i, \mathbf{X}_i)$ are multiplied by the w_i factor, which is inversely proportional to the width of the rail. To prevent distortion of the results due to unreasonable weighting W_{max} is used. This value is the percentile of all w_i values from the training set. This value is around 20 when data augmentation is used.

The m_i factor is used to zero out and ignore all values above the y-limit. Furthermore, it is ensured that the averaging of the loss is conducted exclusively over the relevant segment of the track. This is done by summing all m_i values in the denominator of Equation 4.

$$L_{traj} = \frac{\sum_{i=1}^{H} m_i \times w_i \times L_{rails}(\hat{\mathbf{X}}_i, \mathbf{X}_i)}{\sum_{i=1}^{H} m_i} \tag{4}$$

$$L_{rails} = \sum_{r \in \{left, right\}} SmoothL1(\hat{\mathbf{X}}_{i,r} - \mathbf{X}_{i,r}, \beta_1) \tag{5}$$

$$SmoothL1(x, \beta) = \begin{cases} 0.5x^2/\beta, & \text{if } |x| < \beta \\ |x| - 0.5 * \beta, & \text{otherwise} \end{cases} \tag{6}$$

$$w_i = \min\left(\frac{1}{\mathbf{X}_{i,right} - \mathbf{X}_{i,left}}, W_{max}\right) \tag{7}$$

$$m_i = \begin{cases} 1 & \text{if } i \leq y_{lim} \times H \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Y-Limit Loss L_{ylim} is responsible for calculating the error of the horizon line. For this sub-function also the $SmoothL1$ loss is taken with a $beta_2 = 0.015$. The value of β_2 is chosen to be higher, because of greater inaccuracy of annotations. Due to images from the dataset stemming from low-resolution YouTube videos, it becomes difficult to accurately label the horizon line, especially when the end of the rail track is located at a significant distance.

$$L_{ylim} = SmoothL1(\hat{y_{lim}} - y_{lim}, \beta_2) \tag{9}$$

Figure 23 shows a plot of the Equation 6 because it is an important part of this loss function.

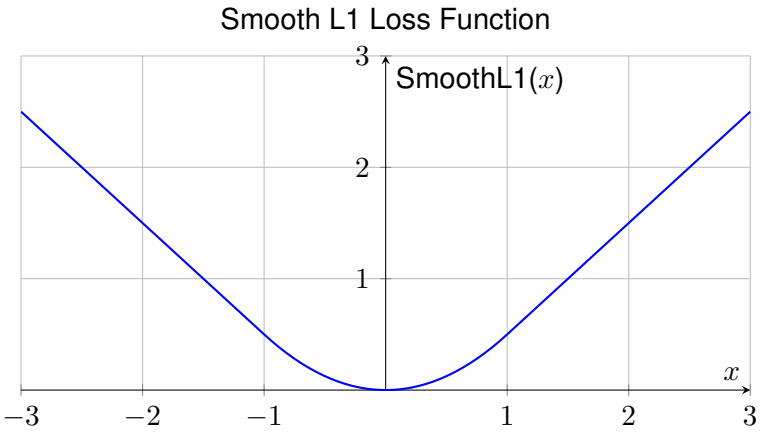


Figure 23: $SmoothL1$ Loss Funktion

5.6.5 Experiments, Results and Comparison other state-of-the-art approaches of TEP-Net

State-of-the-art approaches include two main ideas: a classification and a segmentation approach. To fairly compare the novel regression-based architecture, [1] extends the model with two prediction heads, which support the two state-of-the-art methods. For the segmentation model a U-Net-like [87] decoder architecture is implemented, which outputs a binary mask with dimensions $1 \times 512 \times 512$. A binary Dice loss is used for training. The classification model is inspired by [81]. Therefore the prediction is also described in a grid system with the dimensions $C \times H \times (W + 1)$. With $C = 2$ for the number of rails, $H = 64$ like the number of anchors in the regression model, and $W = 128$ for the width of the grid following the settings from [81]. The $+1$ is for the background class. Consequently, the classification model outputs a $2 \times 64 \times (128 + 1) = 16512$-dimensional vector. A cross-entropy loss is used to train this model [1].

For experiments, the same dataset is utilized. With various preprocessing steps, the data annotation is transformed to fit the specific task of the model. As described in section 5.5 the area between the rails is transformed into a binary mask for segmentation and the grid system is used for classification. To get the most transparent comparison the same backbone is used for the three different models, as illustrated in Figure 24.

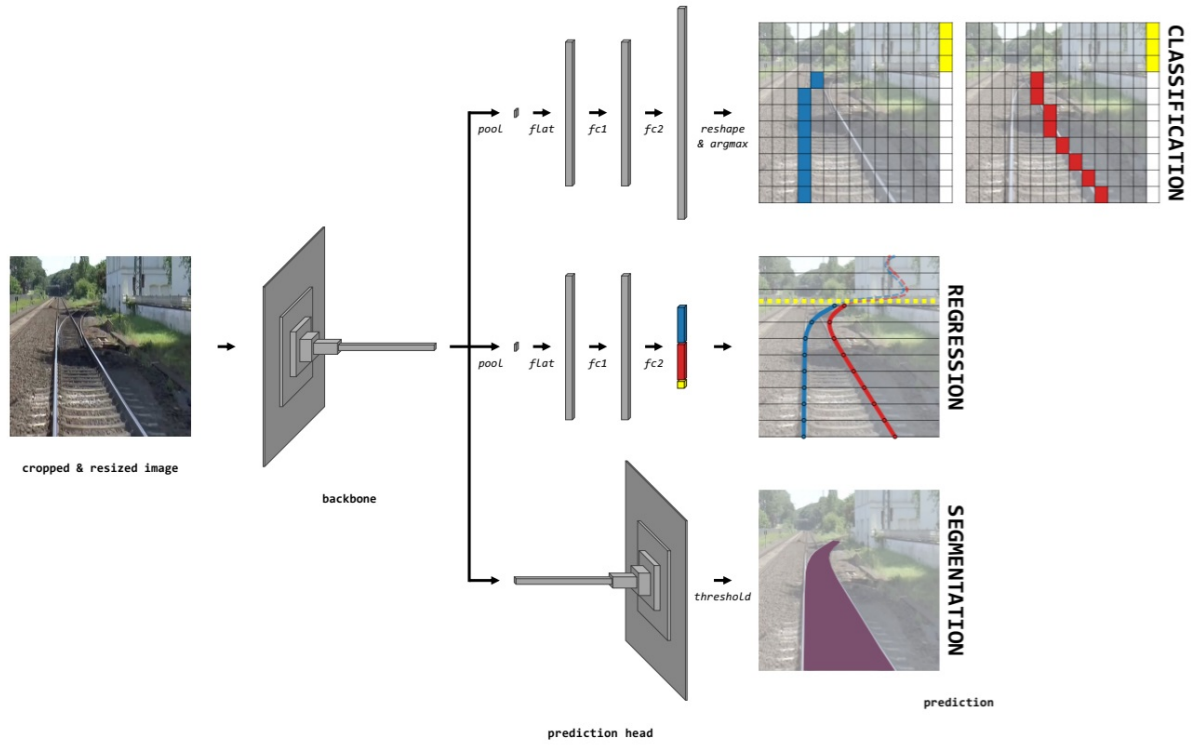


Figure 24: The model architecture is designed to enable a fair comparison between the novel regression model proposed in [1] and other State Of The Art (SOTA) approaches. All models use the same dataset with preprocessing steps to fit annotation to the model task.

Experiments include trainings with different backbones: ResNet18, 34, 50 and EfficientNet B0, B1, B2, B3. Results of [1] show that the segmentation-based approach is the most accurate when it comes to the Intersection Over Union (IoU). However, the difference in IoU performances is within a range of only 1.4 percent. The segmentation model with the EfficientNet-B3 is the best and the classification model with the ResNet18 is the worst performing. The Latency on the other hand shows that the segmentation model is the slowest. Here the regression-based approach outperforms other models. For the rail track prediction application of this work, the latency is of great importance. Additionally, the regression model has lower numbers in parameters and Multiply-Accumulate Operations (MACs), proving to be more lightweight compared to state-of-the-art models. For more detailed results, please refer to [1].

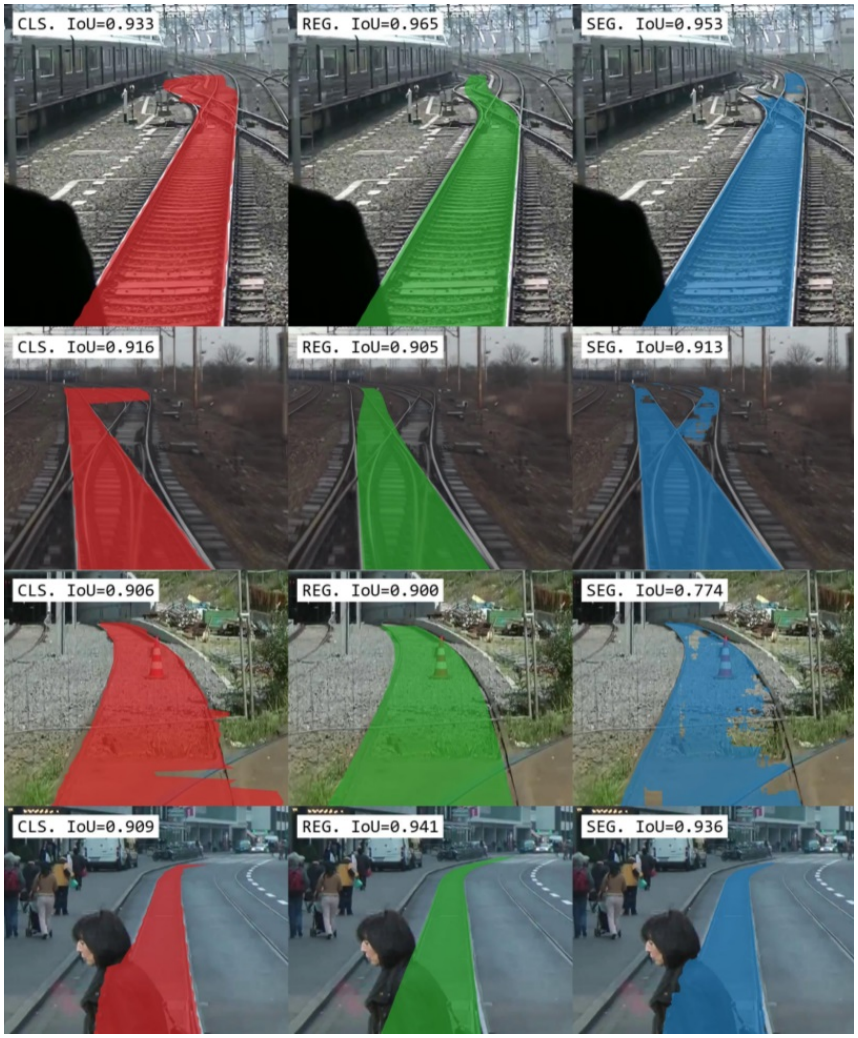


Figure 25: Comparison between classification (CLS), regression (REG) and segmentation (SEG) models with challenging scenes. The ResNet18 backbone is used for this figure.

Since all three model architectures achieve similar IoU scores, performances are compared on individual challenging scenes. To see the differences more clearly the ResNet18 backbone is used because it is the worst performing in terms of accuracy. As visualized in Figure 25, the

correctness of the prediction is negatively impacted when the model becomes unsure of where the rail track continues. However, Figure 25 shows that the regression model is the only one that seems to have no issues with obstructions like in the lower two rows. Additionally, when tracks split at upcoming switches, the novel approach of [1] is the only one that keeps the form of a rail.

This is due to the loss functions of both state-of-the-art models. Both the dice loss and the cross-entropy loss penalize their models in the case of misclassification. However, this penalty does not grow as the distance from the GT increases.

The segmentation and the classification model work with probabilities, either on a pixel-wise or a grid-row level. When segmentation models are confident, their pixel predictions tend toward the extremes of the output range. Under uncertainty, they move closer to the middle which is also the threshold. Similarly, classification models produce one-hot-like distributions when confident, but show more spread-out probabilities across classes when uncertain. Since the regression approach inherently involves continuous values, the model assumes an average among the uncertain possibilities. The concept of distance in the error between prediction and GT is only provided in the regression-based model, but missing in segmentation and classification models. Resulting in a more robust system [1].

5.6.6 Limitation

The main limitation of [1] lies in its single-frame-based model architecture. This model cannot capture temporal context, which becomes problematic when the train encounters a switch. An example scenario is illustrated in Figure 26. Typically, the model effectively predicts the train's path when approaching switches, with all necessary information contained within the frame. However, once the train passes over the switch and the start of the switch is no longer visible, the model cannot determine the continuation of the ego path. Only after a certain duration, does the correct track become identifiable again.

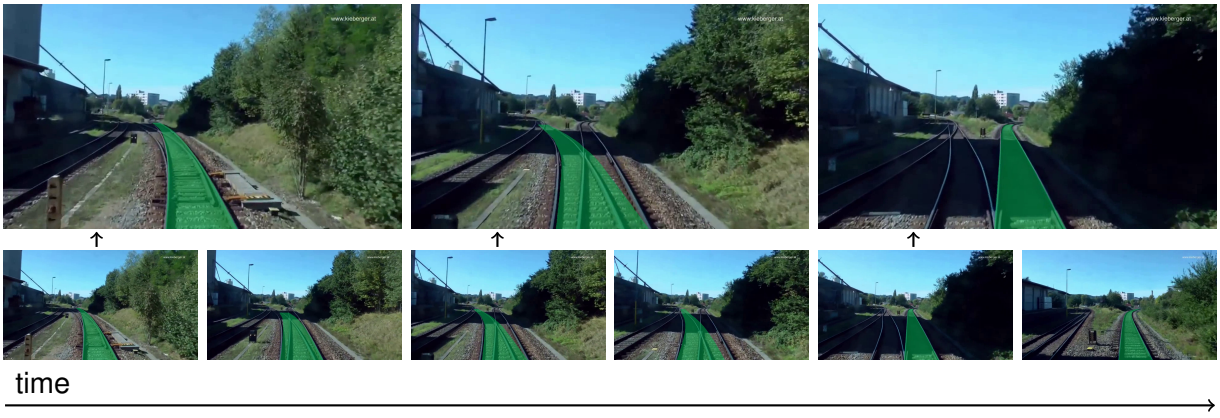


Figure 26: Limitation of TEP-Net [1]. The introduced approach is a single-frame-based model. Therefore, no temporal context can be captured, which leads to uncertainty in prediction when driving over a switch. All images are from [88]. A YouTube video, which is also used by RailSem19. It is ensured that none of the frames are included in the dataset, creating a fair test scenario.

There are two approaches suggested in [1]. The first one includes integrating a confidence score to tell if the model is in a scenario where it is prone to become unreliable. The second suggested approach is more complete. It would encounter the temporal component by implementing a model like a Recurrent Neural Network (RNN), that can capture temporal information. However, [1] states that there is no public temporal dataset available, which fits this task. To the best of the author knowledge, this statement is correct. Therefore, a corresponding dataset must also be created, if this approach is pursued.

6 Methodology

blindtext

6.1 Datasets

blindtext

6.1.1 RailSem19 and used subsets

blindtext

6.1.2 vielleicht: Rail-DB for multi rail setection

blindtext

6.1.3 used annotations

blindtext

6.2 Switch evaluation dataset

blindtext

6.3 labeling task for temporal models

CVAT

Autolabler

blindtext

6.4 Setup used for Training CNNs

For training Convolutional Neural Network (CNN)s a powerful hardware setup is necessary. In this work, one main setup is used to train all models. This work is based on a project proposed and supported by the Institute for Computer Technology (ICT) at the Technical University Vienna (TU), which also provides the necessary resources. The department has a server with two NVIDIA Tesla V100S-PCIE-32GB Graphics Processing Unit (GPU)s [89]. Since the GPUs utilized are equipped with 32 GB of memory and this is sufficient for the needs of this study, all trainings are done on a single GPU. No multi-GPU setup is needed. For this project, CUDA version 12.6 is used, which provides an environment for developing GPU-accelerated applications [90]. The server employs two Intel(R) Xeon(R) Gold 5118 CPUs @ 2.30GHz [91]. These Central Processing Unit (CPU)s have 24 threads and 12 cores each.

All trainings are logged with the "Weights & Biases" [92], a developer platform for training and fine-tuning machine learning models. [92] is utilized for logging configurations and results in this work. The *train loss* and *validation loss* of each training is tracked and visualized in a graph. The *test IoU* and the *best validation loss* are displayed at the end of each training. Additionally, the GPU's power usage and allocated memory are tracked and graphs are plotted. Moreover, "Weights & Biases" [92] assigns a unique name to each training session, a critical feature when starting hundreds of training sessions. All training logs are available at https://wandb.ai/sebiorganization/train-ego-path-detection.

6.5 Measuing Inference on NVIDIA Jetson Device

Due to the nature of this work's use case and potential applications for the system encompassing safety features or preprocessing steps for autonomous trains, the system has to ensure real-time capability when deployed on embedded devices. In more detail, one goal for the train track prediction is to be deployed on an Nvidia Jetson device, because of their high computing power and their low power consumption. Additionally, through NVIDIA's software ecosystem, rapid deployment and latency measurements are possible. Jetson devices are suitable for applications in autonomous systems and computer vision tasks [93]. Furthermore, the NVIDIA Jetson series is specifically built for machine learning applications, because the devices have Deep Learning Accelerator (DLA)s built in. DLAs are tensor processor units designed to accelerate the inference of neuronal networks [94].

6.5.1 Hardware Setup for Measuring Inference

For this study, the NVIDIA Jetson AGX Xavier is chosen because of several reasons. This platform achieves up to 32 TOPS by utilizing a GPU with 512 cores and 64 Tensor cores, which is advantageous for parallel data processing and neural network inference [95]. An additional advantage present the two built-in NVIDIA Deep Learning Accelerator (NVDLA)s of the AGX

Xavier. NVDLAs are NVIDIAs own DLAs. Even though there are more powerful devices like some of the NVIDIA Jetson Orin series, the AGX Xavier is sufficient for this application and cheaper [96]. Additionally, the Xavier has a lower power consumption than the Orin. Finally but yet important is the ease of integration. Since, track prediction is a use case of a company, that most commonly uses the Jetson AGX Xavier platform. Conducting tests on this specific device is appropriate. The technical specifications of the NVIDIA Jetson AGX Xavier are shown in table 5.

AI Performance	32 TOPS
GPU	512-core NVIDIA Volta GPU with 64 Tensor Cores
CPU	8-core NVIDIA Carmel ARM v8.2 64-bit CPU | 8 MB L2 + 4 MB L3
Memory	32 GB 256-Bit LPDDR4x | 136.5 GB/s
Storage	32 GB eMMC 5.1
DL Accelerator	(2x) NVDLA
Power	10 W - 30 W

Table 5: Jetson AGX Xavier technical specifications [95]

6.5.2 Optimizing models with TensorRT

To fully leverage the used hardware platform Jetson AGX Xavier, NVIDIA introduced TensorRT [97]. This ecosystem is developed to allow faster inference times when deploying deep learning models. Also, the baseline paper [1] demonstrates through latency measurements that TensorRT consistently outperforms PyTorch in the context of speed. TensorRT is approximately six times faster than PyTorch, which aligns with the claims made by NVIDIA [1] [97]. Consequently, all latency measurements in this study are conducted using TensorRT. This framework optimizes inference with methods like quantization, layer and tensor fusion, and kernel tuning [97]. This can be done for various types of NVIDIA GPUs.

Quantization can optimize an already trained model. This technique shows a small reduction in accuracy but minimizes latency significantly. While PyTorch uses PF32 for the inference of its standard models, TensorRT allows to use GPUs and Tensor Processing Unit (TPU)s to their maximum capacity by permitting FP8, INT8, and INT4.

Layer and Tensor fusion are used from TensorRT for further optimizing inference. Often specific layer sequences include two consecutive layers that can be mathematically combined into a single layer. Resulting in a reduction of unessential computations.

Kernal tuning is a process that seeks the optimal configuration of available kernels in the Jetson device. The selected kernels depend on the specific machine-learning application and the used device. In more detail, the model is executed on the device several times using different CUDA kernels in each run and the best combination is utilized. Since, Kernel tuning is an iterative process it usually takes a couple of minutes even with compact models.

TensorRT engine from PyTorch model

Since TensorRT does not support PyTorch models, a workaround has to be made. First, the PyTorch models are converted into an Open Neural Network Exchange (ONNX) format [98]. After that, the `.onnx` files are optimized by TensorRT with the techniques mentioned before.

ONNX is an approach for easier access to hardware optimization and to make interoperability possible [98]. Often machine learning applications are locked in the framework they are developed in, which can present some hurdles. The ONNX format aims for a standardized representation of machine learning models and is therefore commonly used in the community. Once converted to ONNX, a model utilizes standard data types and a set of built-in operators. The ONNX format version, known as the "opset", defines which operators are used [98]. Therefore the TensorRT version must be compatible with the ONNX opset number.

The TensorRT version installed on the Jetson AGX Xavier must support the used ONNX opset. Therefore in this work, the opset 11 is used for all model exports.

```
# Export the model to ONNX
torch.onnx.export(
    model,                          # Model to be exported
    input_tensor,                   # Input to the model
    "onnx_file_path/model.onnx",    # Output file path
    opset_version=11,               # ONNX version to export the model to
    export_params=True,             # Store the trained parameter
                                    # weights inside the model file
)
```

Listing 2: Exporting a PyTorch model to ONNX format

In this work, all PyTorch models are converted to `.onnx` files with the built-in torch exporter. Listing 2 shows that the conversion is done with a single `torch.onnx.export()` python line [99]. After an ONNX model format is created it can be optimized by TensorRT. This can be executed with the `trtexec` console application.

```
trtexec –onnx=model.onnx –saveEngine=model.engine
```

This command provides an example, in which the model.onnx is optimized with TensorRT techniques mentioned above. Furthermore, the `model.engine` is saved and can be executed from now on without the need to create it again. After a couple of minutes, TensorRT outputs a performance summary with many different latencies, like the min, max, mean, or median. Of these values, the median inference time is considered as the final latency rather than the mean, because it is more robust against outliers.

7 Experiments

Etwas Text... Hier kommen noch einige Abkürzunge vor zum Beispiel ABC,WWW und ROFL.

7.1 Ensemble Learning Approach

blindtext

7.2 improved TEP-Net

blindtext

7.2.1 Autocrop

blindtext

7.2.2 Backbones

blindtext

7.2.3 Pooling Layers

blindtext

7.2.4 Prediction Heads

blindtext

7.2.5 Sliding Window Approach

blindtext

7.2.6 Temporal Models

blindtext

7.2.7 Erste Überschrift Tiefe 3 (subsection)

blindtext

7.2.8 Erste Überschrift Tiefe 3 (subsection)

blindtext

Erste Überschrift Tiefe 4 (subsubsection)

blindtext

8 Results

Etwas Text... Hier kommen noch einige Abkürzunge vor zum Beispiel ABC,WWW und ROFL.

8.1 Ensemble Learning Approach

blindtext

8.2 improved TEP-Net

blindtext

8.2.1 Autocrop

blindtext

8.2.2 Backbones

blindtext

8.2.3 Pooling Layers

blindtext

8.2.4 Prediction Heads

blindtext

8.2.5 Sliding Window Approach

blindtext

8.2.6 Temporal Models

blindtext

8.3 Erste Überschrift Tiefe 2 (section)

blindtext

8.3.1 Erste Überschrift Tiefe 3 (subsection)

blindtext

Erste Überschrift Tiefe 4 (subsubsection)

blindtext

9 Discussion

Etwas Text... Hier kommen noch einige Abkürzunge vor zum Beispiel ABC,WWW und ROFL.

9.1 Erste Überschrift Tiefe 2 (section)

blindtext

9.1.1 Erste Überschrift Tiefe 3 (subsection)

blindtext

Erste Überschrift Tiefe 4 (subsubsection)

blindtext

10 Conclusion and Outlook

Etwas Text... Hier kommen noch einige Abkürzunge vor zum Beispiel ABC,WWW und ROFL.

10.1 Erste Überschrift Tiefe 2 (section)

blindtext

10.1.1 Erste Überschrift Tiefe 3 (subsection)

blindtext

Erste Überschrift Tiefe 4 (subsubsection)

blindtext

Bibliography

[1] T. Laurent, *Train ego-path detection on railway tracks using end-to-end deep learning*, 2024. arXiv: 2403.13094 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2403.13094.

[2] H. Kopka, *LaTeX, Band 1: Einführung*, 3rd ed. München: Pearson Studium, 2005.

[3] ——, *LaTeX, Band 1: Einführung*, 3rd ed. München: Pearson Studium, 2005. [Online]. Available: http://www.pearson-studium.de (visited on 07/06/2011).

[4] M. Goossens, F. Mittelbach, and A. Samarin, *Der LaTeX Begleiter*. Bonn: Addison-Wesley Deutschland, 2002.

[5] S. Teschl, K. M. Göschka, and G. Essl, *Leitfaden zur Verfassung einer Bachelorarbeit oder Master Thesis*, FH Technikum Wien, 2014. [Online]. Available: www.technikum-wien.at (visited on 08/04/2014).

[6] M. Humenberger, D. Hartermann, and W. Kubinger, "Evaluation of stereo matching systems for real world applications using structured light for ground truth estimation," in *Proceedings of the Tenth IAPR Conference on Machine Vision Applications (MVA2007)*, Tokyo, Japan: MVA Conference Committee, May 16, 2007, pp. 433–436.

[7] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1180–1202, 2010.

[8] C. Zinner, W. Kubinger, and R. Isaacs, "Pfelib: A performance primitives library for embedded vision," *EURASIP Journal on Embedded Systems*, vol. 2007, pp. 1–14, 2007. [Online]. Available: http://downloads.hindawi.com/journals/es/2007/049051.pdf (visited on 07/06/2011).

[9] H. Hemetsberger, *Ait stereo sensor im einsatz während der darpa urban challenge 2007*, AIT Austrian Institute of Technology, 2007.

[10] Siemens Automation Technology, *Simatic*, 2011. [Online]. Available: http://www.automation.siemens.com/mcms/topics/de/simatic/Seiten/Default.aspx (visited on 07/06/2011).

[11] ——, *Simatic*, [Online] Verfügbar unter: <http://www.automation.siemens.com/mcms/topics/de/simatic/Seiten/Default.aspx> [Zugang am 17.10.2014], 2014.

[12] International Standards Office, *Iso 690 – information and documentation: Bibliographical references: Electronic documents*, Genf: International Standards Office, 1998.

[13] Atmel Corporation, *Atmel atmega16 – 8-bit microcontroller with 16k bytes in-system programmable flash*, San Jose, United States: Atmel Corporation, 2011. [Online]. Available: http://www.atmel.com/dyn/resources/prod%5C_documents/doc2466.pdf (visited on 07/06/2011).

[14] M. Humenberger, *Real-time stereo matching for embedded systems in robotic applications*, Wien: Technische Universität Wien, Fakultät für Elektrotechnik und Informationstechnik, 2011.

[15] J. Pohn, *Condition monitoring systeme für die zustandorientierte instandhaltung von windkraftanlagen*, Wien: FH Technikum Wien, Masterstudiengang Innovations- und Technologiemanagement, 2010.

[16] Statista-Austria. (2024). Anzahl der beförderten personen im schienenpersonenverkehr in österreich von 2009 bis 2023 (in millionen) [graph], [Online]. Available: https://de.statista.com/statistik/daten/studie/296524/umfrage/schienenpersonenverkehr-der-eisenbahn-in-oesterreich/ (visited on 11/05/2024).

[17] Wiener-Stadtwerke. (2024). Anzahl der fahrgäste der wiener linien von 2006 bis 2023 (in millionen) [graph], [Online]. Available: https://de.statista.com/statistik/daten/studie/714378/umfrage/fahrgaeste-der-wiener-linien/ (visited on 11/05/2024).

[18] Österreichische-Bundesbahnen. (2024). Anzahl der fahrgäste der österreichischen bundesbahnen (öbb) von 2012 bis 2023 (in millionen) [graph], [Online]. Available: https://de.statista.com/statistik/daten/studie/299369/umfrage/fahrgaeste-der-oesterreichischen-bundesbahnen/ (visited on 11/05/2024).

[19] Weninger, *Verkehrstatistik 2022*. 2022. [Online]. Available: https://www.statistik.at/fileadmin/user_upload/Verkehr-2022-barr.pdf (visited on 11/05/2024).

[20] red, *Zug entgleist: 40 passagiere gerettet*, 2024. [Online]. Available: https://noe.orf.at/stories/3260056/ (visited on 11/05/2024).

[21] Stefan Schwarzwald-Sailer, *Tödlicher zugsunfall bringt öbb in erklärungsnot*, 2024. [Online]. Available: https://noe.orf.at/stories/3259860/ (visited on 11/05/2024).

[22] red, *Mehrere tote bei zugsunglück in tschechien*, 2024. [Online]. Available: https://orf.at/stories/3359814/ (visited on 11/05/2024).

[23] Fraunhofer Institute for Cognitive Systems IKS, *Autonomous driving - fraunhofer iks*, 2024. [Online]. Available: https://www.iks.fraunhofer.de/en/topics/autonomous-driving.html (visited on 11/05/2024).

[24] KPMG. (2020). 2020 autonomous vehicles readiness index, [Online]. Available: https://assets.kpmg.com/content/dam/kpmg/xx/pdf/2020/07/2020-autonomous-vehicles-readiness-index.pdf (visited on 11/05/2024).

[25] B. Impey. (2024). Entwicklungsstand im bereich autonomer mobilität nach ländern weltweit nach autonomous vehicle readiness index[1] (stand: 2020 und vergleich zum vorjahr) [graph], [Online]. Available: https://de.statista.com/statistik/daten/studie/1113863/umfrage/entwicklungsstand-im-bereich-autonomer-mobilitaet-nach-laendern-weltweit/ (visited on 11/05/2024).

[26] O. Zendel, M. Murschitz, M. Zeilinger, D. Steininger, S. Abbasi, and C. Beleznai, "Railsem19: A dataset for semantic rail scene understanding," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, 2019, pp. 1221–1229, ISBN: 978-1-7281-2506-0. DOI: 10.1109/CVPRW.2019.00161.

[27] Y. Wang, L. Wang, Y. H. Hu, and J. Qiu, "Railnet: A segmentation network for railroad detection," *IEEE Access*, vol. 7, pp. 143 772–143 779, 2019. DOI: 10.1109/ACCESS.2019.2945633.

[28] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, *Panoptic segmentation*, 2019. arXiv: 1801.00868 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1801.00868.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[30] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[31] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023. DOI: 10.1109/JPROC.2023.3238524.

[32] R. R, I. Fatima, and L. A. Prasad, "A survey on real-time object detection algorithms," in *2023 International Conference on Advances in Electronics, Communication, Computing and Intelligent Information Systems (ICAECIS)*, 2023, pp. 548–553. DOI: 10.1109/ICAECIS58353.2023.10170349.

[33] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019. DOI: 10.1109/ACCESS.2019.2939201.

[34] G. Singh, S. Tiwari, and J. Singh, "Real time object detection using neural networks: A comprehensive survey," in *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 2023, pp. 1281–1286. DOI: 10.1109/ICAIS56108.2023.10073826.

[35] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[36] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. DOI: 10.1109/TPAMI.2016.2577031.

[37] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[39] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[40] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. arXiv: 1804.02767 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1804.02767.

[41] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: 2004.10934 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2004.10934.

[42] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46448-0.

[44] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. arXiv: 2207.02696 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2207.02696.

[45] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, *Yolov9: Learning what you want to learn using programmable gradient information*, 2024. arXiv: 2402.13616 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2402.13616.

[46] R. R, I. Fatima, and L. A. Prasad, “A survey on real-time object detection algorithms,” in *2023 International Conference on Advances in Electronics, Communication, Computing and Intelligent Information Systems (ICAECIS)*, 2023, pp. 548–553. DOI: 10.1109/ICAECIS58353.2023.10170349.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[48] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: https://www.image-net.org/challenges/LSVRC/2012/index.php (visited on 11/19/2024).

[49] A. Younesi, M. Ansari, M. Fazli, A. Ejlali, M. Shafique, and J. Henkel, "A comprehensive survey of convolutions in deep learning: Applications, challenges, and future trends," *IEEE Access*, vol. 12, pp. 41 180–41 218, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267897956.

[50] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.1556.

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, " Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298594.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[53] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[54] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: 1704.04861 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1704.04861.

[55] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[56] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, " Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2019, pp. 1314–1324. DOI: 10.1109/ICCV.2019.00140. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2019.00140.

[57] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.

[58] PyTorch, *Resnet*, 2024. [Online]. Available: https://pytorch.org/vision/main/models/resnet.html (visited on 11/20/2024).

[59] ——, *Densenet*, 2024. [Online]. Available: https://pytorch.org/vision/main/models/densenet.html (visited on 11/20/2024).

[60] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, *Squeeze-and-excitation networks*, 2019. arXiv: 1709.01507 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1709.01507.

[61] PyTorch, *Mobilenet v3*, 2024. [Online]. Available: https://pytorch.org/vision/main/models/mobilenetv3.html (visited on 11/20/2024).

[62] ——, *Efficientnet*, 2024. [Online]. Available: https://pytorch.org/vision/main/models/efficientnet.html (visited on 11/20/2024).

[63] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," Toronto, ON, Canada, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744.

[64] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, pp. 98–136, 2015.

[65] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:14113767.

[66] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Apr. 2015, ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0816-y.

[67] A. Kuznetsova, H. Rom, N. G. Alldrin, J. R. R. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The open images dataset v4," *International Journal of Computer Vision*, vol. 128, pp. 1956–1981, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53296866.

[68] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, " The Cityscapes Dataset for Semantic Urban Scene Understanding," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2016, pp. 3213–3223. DOI: 10.1109/CVPR.2016.350. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.350.

[69] G. Neuhold, T. Ollmann, S. R. Bulò, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5000–5009. DOI: 10.1109/ICCV.2017.534.

[70] H. Caesar, J. Uijlings, and V. Ferrari, "Coco-stuff: Thing and stuff classes in context," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018. DOI: 10.1109/cvpr.2018.00132. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2018.00132.

[71] H. Abu Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *International Journal of Computer Vision*, vol. 126, pp. 961–972, 2018. DOI: https://doi.org/10.1007/s11263-018-1070-x.

[72] M. A. Hadded, A. Mahtani, S. Ambellouis, J. Boonaert, and H. Wannous, "Application of rail segmentation in the monitoring of autonomous train's frontal environment," in *Pattern Recognition and Artificial Intelligence*, ser. Lecture Notes in Computer Science, M. El Yacoubi, E. Granger, P. C. Yuen, U. Pal, and N. Vincent, Eds., vol. 13363, Cham: Springer International Publishing, 2022, pp. 185–197, ISBN: 978-3-031-09036-3. DOI: 10.1007/978-3-031-09037-0{\textunderscore}16.

[73] Z. Zhang, S. Yu, S. Yang, Y. Zhou, and B. Zhao, *Rail-5k: A real-world dataset for rail surface defects detection*, 2021. arXiv: 2106.14366 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2106.14366.

[74] S. Ma, K. Song, M. Niu, *et al.*, "Cross-scale fusion and domain adversarial network for generalizable rail surface defect segmentation on unseen datasets," *Journal of Intelligent Manufacturing*, vol. 35, pp. 367–386, 2024. DOI: 10.1007/s10845-022-02051-7.

[75] X. Huang, K. Ye, Z. Fang, Y. Xie, X. Ma, J. Ji, and Q. Wu, "Research on grooved rail garbage identification algorithm based on improved yolov3," *Journal of Physics: Conference Series*, vol. 1827, no. 1, p. 012 185, 2021. DOI: 10.1088/1742-6596/1827/1/012185. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1827/1/012185.

[76] J. Harb, N. R'eb'ena, R. Chosidow, G. Roblin, R. Potarusov, and H. Hajri, "Frsign: A large-scale traffic light dataset for autonomous trains," *ArXiv*, vol. abs/2002.05665, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211096970.

[77] P. Leibner, F. Hampel, and C. Schindler, "Gerald: A novel dataset for the detection of german mainline railway signals," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 237, no. 10, pp. 1332–1342, 2023.

[78] H. Yuan, Z. Mei, Y. Chen, W. Niu, and C. Wu, "Railvid: A dataset for rail environment semantic," *ICONS*, vol. 2022, 17th, 2022.

[79] A. Zouaoui, A. Mahtani, M. A. Hadded, S. Ambellouis, J. Boonaert, and H. Wannous, "Railset: A unique dataset for railway anomaly detection," in *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, vol. Five, 2022, pp. 1–6. DOI: 10.1109/IPAS55744.2022.10052883.

[80] M. A. Hadded, A. Mahtani, S. Ambellouis, J. Boonaert, and H. Wannous, “Application of rail segmentation in the monitoring of autonomous train’s frontal environment,” in *International Conference on Pattern Recognition and Artificial Intelligence*, Springer, 2022, pp. 185–197.

[81] X. Li and X. Peng, “Rail detection: An efficient row-based network and a new benchmark,” in *Proceedings of the 30th ACM International Conference on Multimedia*, 2022, pp. 6455–6463.

[82] S. Lee, *Rail-detection*, GitHub repository, 2022. [Online]. Available: https://github.com/Sampson-Lee/Rail-Detection.

[83] R. Tilly, P. Neumaier, K. Schwalbe, P. Klasek, R. Tagiew, P. Denzler, T. Klockau, M. Boekhoff, and M. Köppel, *Osdar23: Open sensor data for rail 2023*, de, 2023. DOI: 10.57806/9MV146R0. [Online]. Available: https://data.fid-move.de/dataset/3d7e7406-639f-49f6-bbca-caac511b4032.

[84] PyTorch, *Colorjitter*, 2024. [Online]. Available: https://pytorch.org/vision/main/generated/torchvision.transforms.ColorJitter.html (visited on 11/12/2024).

[85] ——, *Conv2d*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html (visited on 11/12/2024).

[86] ——, *Linear*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html (visited on 11/12/2024).

[87] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4. DOI: https://doi.org/10.1007/978-3-319-24574-4_28.

[88] W. YouTube, *Austria ried im innkreis scharding*, 2024. [Online]. Available: https://www.youtube.com/watch?v=Vj9lO19kGLg&t=1224s (visited on 11/12/2024).

[89] NVIDIA Corporation, *Nvidia v100 gpu architecture data sheet*, 2024. [Online]. Available: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf (visited on 11/07/2024).

[90] ——, *Cuda toolkit*, 2024. [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on 11/07/2024).

[91] Intel Corporation, *Intel® xeon® gold 5118 processor data sheet*, 2024. [Online]. Available: https://ark.intel.com/content/www/de/de/ark/products/120473/intel-xeon-gold-5118-processor-16-5m-cache-2-30-ghz.html (visited on 11/07/2024).

[92] Weights & Biases, *Weights & biases*, 2024. [Online]. Available: https://wandb.ai/site (visited on 11/07/2024).

[93] NVIDIA Corporation, *Nvidia jetson for next-generation robotics*, 2024. [Online]. Available: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/ (visited on 11/08/2024).

[94] ——, *Deep learning accelerator (dla)*, 2024. [Online]. Available: https://developer.nvidia.com/deep-learning-accelerator (visited on 11/08/2024).

[95] ——, *Nvidia jetson xavier technical specifications*, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/ (visited on 11/07/2024).

[96] ——, *Buy the latest jetson products*, 2024. [Online]. Available: https://developer.nvidia.com/buy-jetson?product=all&location=DE (visited on 11/08/2024).

[97] ——, *Nvidia tensorrt*, 2024. [Online]. Available: https://developer.nvidia.com/tensorrt (visited on 11/08/2024).

[98] ONNX, *Open neural network exchange*, 2024. [Online]. Available: https://onnx.ai/ (visited on 11/08/2024).

[99] PyTorch, *Torch.onnx*, 2024. [Online]. Available: https://pytorch.org/docs/stable/onnx.html (visited on 11/08/2024).

List of Figures

Figure 1 Beispiel für die Beschriftung eines Buchrückens. 2
Figure 2 2. Beispiel für die Beschriftung eines Buchrückens. 2
Figure 3 Train accidets in **(a)** Austria and **(b)** the Czech Republic, where misinformation of switches resulted in deaths. 4
Figure 4 Example of a diverging rail track at a switch. The trains path is marked in green. The path the train cannot take is marked in red. This path is be obstructed and would be unsafe [1]. *rail detection*-output: green and red track; *rail track prediction*-output: green track 5
Figure 5 The most common applications in vision tasks that are supported by neural networks. Shown are GT examples for each usage domain. The input image is visualized in (a) without the added text in the top right corner [28]. 7
Figure 6 ResNet residual block [52] 16
Figure 7 DenseNet dense block [53] 17
Figure 8 MobileNet V2 and V3 Blocks [56]: **(a)** MobileNetV2 bottleneck block, **(b)** MobileNetV3 bottleneck block with added Squeeze-and-Excite block 18
Figure 9 Systematic visualisation of scaling parameters of a network [57] 19
Figure 10 RailSem19 dataset examples. First row raw images. Second row dense GT [26]. 23
Figure 11 Example images and GT of RailVID dataset [78] 23
Figure 12 RailSet-Seg example with annotations [79] [80]: **(a)** raw-image, **(b)** rail class, **(c)** rail and rail-track class 24
Figure 13 RSDS example with annotation [27]: **(a)** raw-image, **(b)** ground truth 25
Figure 14 Rail-DB [81] images with annotations in different conditions 26
Figure 15 OSDaR23 data from all different sensors [83] 27
Figure 16 OSDaR23 annotated scene [83] 28
Figure 17 OSDaR23 labels skaliert 29
Figure 18 Examples images from RailSem19, which are not included in the TEP-Net dataset due to unclear circumstances about the trains direction. [1] 30
Figure 19 TEP-Net dataset example images with annotation [1] 30
Figure 20 Data augmentation of TEP-Net [1], including the image horizontal flips, the color variations and the cropping mechanism. 33
Figure 21 Example of perspective shifts with three different scenarios: left curve, right curve and straight rail [1]. 34

Figure 22 TEP-Net model architecture[1]. The input of the model is a cropped and resized image and the output of the model are the x-values for the left and right rail on each anchor line plus an additional value for the y-limit. 35
Figure 23 $SmoothL1$ Loss Funktion 37
Figure 24 The model architecture is designed to enable a fair comparison between the novel regression model proposed in [1] and other SOTA approaches. All models use the same dataset with preprocessing steps to fit annotation to the model task. 38
Figure 25 Comparison between classification (CLS), regression (REG) and segmentation (SEG) models with challenging scenes. The ResNet18 backbone is used for this figure. 39
Figure 26 Limitation of TEP-Net [1]. The introduced approach is a single-frame-based model. Therefore, no temporal context can be captured, which leads to uncertainty in prediction when driving over a switch. All images are from [88]. A YouTube video, which is also used by RailSem19. It is ensured that none of the frames are included in the dataset, creating a fair test scenario. 41

List of Tables

Table 1 Semesterplan der Lehrveranstaltung „Angewandte Mathematik“. 2
Table 2 2. Semesterplan der Lehrveranstaltung „Angewandte Mathematik“. 2
Table 3 Datasets for Classification 20
Table 4 Datasets for Semantic Segmentation 21
Table 5 Jetson AGX Xavier technical specifications [95] 44

Quellcodeverzeichnis

1 Hello-World 3
2 Exporting a PyTorch model to ONNX format 45

Abkürzungsverzeichnis

ABC Alphabet

WWW world wide web

ROFL Rolling on floor laughing

CV Computer Vision

RGB Red Green Blue

RSDS Railroad Segmentation Dataset

IR Infrared

TEP Train Ego Path

GT Ground Truth

FOV Field Of View

CNN Convolutional Neural Network

ICT Institute for Computer Technology

TU Technical University Vienna

GPU Graphics Processing Unit

IoU Intersection Over Union

CPU Central Processing Unit

TPU Tensor Processing Unit

ONNX Open Neural Network Exchange

DLA Deep Learning Accelerator

NVDLA NVIDIA Deep Learning Accelerator

ROI Region of Interest

RNN Recurrent Neural Network

MACs Multiply-Accumulate Operations

SOTA State Of The Art

YOLO You Only Look Once

RCNN Regions with CNN features

SSD Single Shot MultiBox Detector

PGI Programmable Gradient Information

GELAN Generalized Efficient Layer Aggregation Network

FPN Feature Pyramid Networks

ILSVRC ImageNet Large Scale Visual Recognition Challenge

VGGNet Visual Geometry Group Network

A Anhang A

B Anhang B