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

Robust Rail Track Prediction with Temporal Deep Learning

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, December 3, 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, December 3, 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
5.4 Line Detection . . . 13
5.5 Baseline Paper . . . 16
5.5.1 Comparison other state-of-the-art approaches . . . 17
5.5.2 Limitation . . . 19
5.6 Network architectures . . . 20
5.6.1 ResNets . . . 21
5.6.2 DenseNets . . . 21
5.6.3 MobileNetV3 . . . 22
5.6.4 EfficientNets . . . 23
5.7 Temporal Model Architectures . . . 25
5.8 Datasets . . . 27

6 Methodology 36
6.1 Models . . . 36
6.1.1 Object Detection Models . . . 36
6.1.2 TEP-Net Model . . . 37

6.1.3 Improvements to the TEP-Net Model . . . 38
6.1.4 Temporal Models . . . 42
6.2 Data handling and Training process . . . 45
6.3 Datasets . . . 46
6.3.1 RailSem19 and subsets for training object detection models . . . 46
6.3.2 TEP annotations . . . 47
6.3.3 Switch evaluation dataset . . . 47
6.3.4 Temporal Dataset . . . 49
6.4 Data augmentation . . . 50
6.4.1 Data augmentation for Training single-frame-based models . . . 50
6.4.2 Data augmentation for Training temporal models . . . 51
6.4.3 Cropping in inference with TEP's Autocrop . . . 52
6.4.4 Improved Autocrop for inference . . . 53
6.5 Loss function . . . 54
6.6 Setup used for Training CNNs . . . 56
6.7 Measuing Inference on NVIDIA Jetson Device . . . 57
6.7.1 Hardware Setup for Measuring Inference . . . 57
6.7.2 Optimizing models with TensorRT . . . 58

7 Experiments **61**
7.1 Object detection Experiments . . . 61
7.2 improved TEP-Net . . . 62
7.2.1 Autocrop . . . 62
7.2.2 Backbones . . . 62
7.2.3 Pooling Layers . . . 62
7.2.4 Prediction Heads . . . 62
7.2.5 Sliding Window Approach . . . 62
7.2.6 Temporal Models . . . 62
7.2.7 Erste Überschrift Tiefe 3 (subsection) . . . 63
7.2.8 Erste Überschrift Tiefe 3 (subsection) . . . 63

8 Results **64**
8.1 Ensemble Learning Approach . . . 64
8.2 improved TEP-Net . . . 64
8.2.1 Autocrop . . . 64
8.2.2 Backbones . . . 64
8.2.3 Pooling Layers . . . 64
8.2.4 Prediction Heads . . . 64
8.2.5 Sliding Window Approach . . . 64
8.2.6 Temporal Models . . . 64

8.3 Erste Überschrift Tiefe 2 (section) 65
8.3.1 Erste Überschrift Tiefe 3 (subsection) 65

9 Discussion **66**
9.1 Erste Überschrift Tiefe 2 (section) 66
9.1.1 Erste Überschrift Tiefe 3 (subsection) 66

10 Conclusion and Outlook **67**
10.1 Erste Überschrift Tiefe 2 (section) 67
10.1.1 Erste Überschrift Tiefe 3 (subsection) 67

Bibliography **68**

List of Figures **81**

List of Tables **84**

Quellcodeverzeichnis **85**

Abkürzungsverzeichnis **86**

A Anhang A **88**

B Anhang B **89**

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 16, in the right two images of figure 17 and in figure 18. 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.

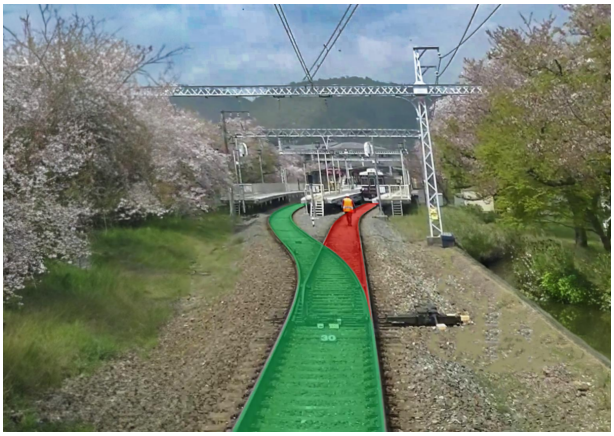


Figure 4: Example of a diverging rail track at a switch. The trains path is marked in green. The path the train cannot take is marked in red. This path is be obstructed and would be unsafe [1]. *rail detection*-output: green and red track; *rail track prediction*-output: green track

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, Train Ego Path (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 Recurrent Neural Network (RNN)s, 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.

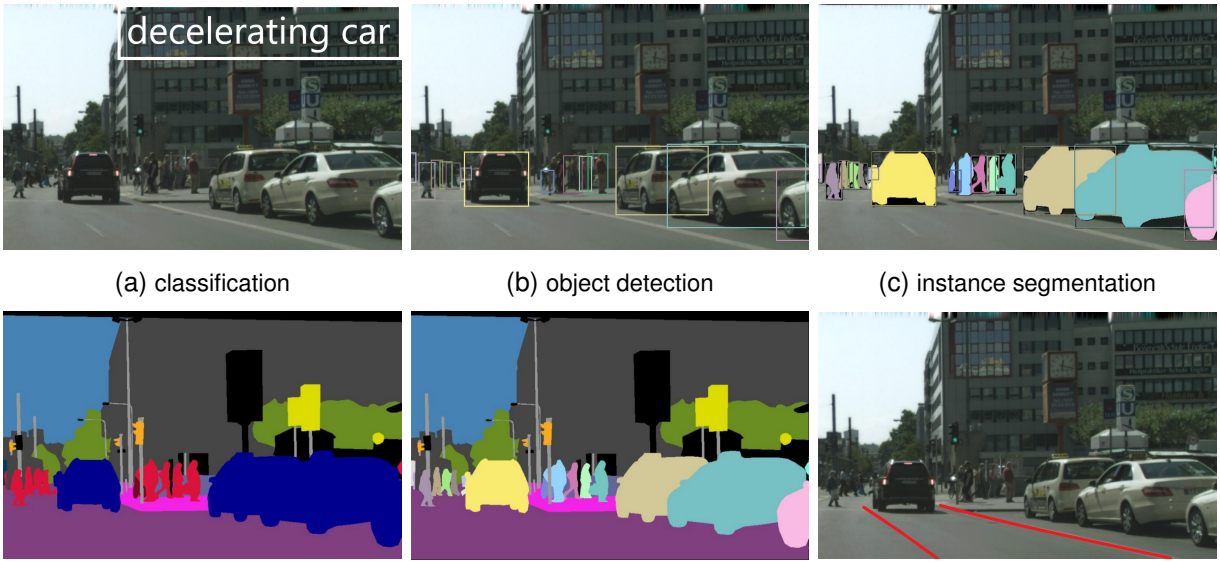
5 State of the Art describes different approaches, to how the train track prediction problem can be solved. In this section, thorough research is done, which gives an overview of various approaches, models, and fitting datasets. Additionally, a couple of papers that could serve as a baseline are discussed 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 output bounding boxes and can detect 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

Semantic Segmentation models are used to create a mask of an image, in which every pixel is assigned to a class. This method can be used to segment rails or rail tracks to obtain the direction of the train.

The first approach to filter out the rail tracks in front of the train was proposed by [47] in 2018. A SegNet [48] inspired network for semantic segmentation is extended with mixed pooling [49] and Atrous Spatial Pyramid Pooling (ASPP) from DeepLab [50]. 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 [27] introduced the RailNet architecture. This model uses a ResNet-50 [51] backbone and a fully convolutional network [52] to segment the rail tracks. The network is designed in a pyramid structure [38], in which features of every ResNet stage are summed and up-sampled. This combines low and high-level features, resulting in an enhanced performance. [27] reports higher accuracy than state-of-the-art segmentation models at the time with speeds up to 20 Frames Per Second (FPS). Tests are made on the introduced Railroad Segmentation Dataset (RSDS) dataset further described in section 5.8.

In 2019 RailSem19 [26] introduced the first publicly available dataset for the rail domain. This dataset also includes annotations for semantic segmentation tasks and experimented models like FRRNB [53]. This dataset is widely used in the community when working in the rail domain. [54] already uses a subset of RailSem19 in 2020 and proposed another model named RailNet that uses a VGG16 [55] backbone and an Information Aggregation Module. Only rails are predicted, other annotations of RailSem19 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. [56] uses four subsets of RailSem19 with 2, 3, 4, or all 19 classes to train the U-Net [57] architecture. Additionally, A cropping method is implemented to account for the difference in resolutions of RailSem19's images and the U-Net recommended one. [58] used the RailSet dataset [59], which is an extension of RailSem19 for segmentation and anomaly detection tasks. For details of the RailSet dataset please refer to section 5.8. U-Net [57] and FRNN [53] are trained with horizontal flips and zooming for the data augmentation.

In 2022 [60] proposed the RailVID dataset, which consists of infrared images instead of Red Green Blue (RGB) data to improve the system's abilities in challenging situations such as the absence of ambient light. The dataset is described in section 5.8 in more detail. After collecting data, [60] experimented with widely used Convolutional Neural Network (CNN)s for semantic segmentations, like Context Grided Network (CGNet) [61], DeepLabv3+ [62], and Bilateral Semantic segmentation Net (BiSeNet) [63]. 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 [64] 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 [65]. Additionally, a port processing algorithm is utilized that is based on sliding window detection. [64] 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. [66] utilizes a BiSeNet [63] architecture for segmenting rails, which is tailored for the anomaly detection task. Therefore accuracy is preferred over speed. While this network can detect small objects on the track, the system is not real-time capable.

[67] is the first to consider a distinction between the trains own and adjacent tracks. Consequently, the concept of "possible tracks" is introduced, which are all paths the train could follow under the assumption that the state of switches cannot be determined. Figure 6 visualizes this concept. For detecting only the green tracks a U-Net-inspired [57] semantic segmentation network is proposed, which incorporates a rule-based post-processing algorithm. The network combines a ResNet-34 [51] backbone and includes connections to the upsampling blocks. At the deepest level, a Spatial Pyramid Pooling (SPP) [68] and on the skip connections squeeze-and-excitation blocks [69] are used. Tests on RailSem19, show that it outperforms the first proposed RailNet [27] in accuracy with a speed of 20 FPS.

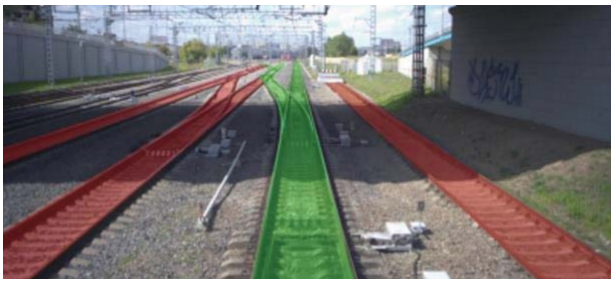


Figure 6: Track segmentation in front of the train. Red are all adjacent tracks and green are possible tracks [67].

In 2023 [70] further investigated the task of possible tracks and introduced Track Point Extraction and Association Network (TPE-Net). It consists of a DenseNet-inspired [71] 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 fitting polynomials. Because of the complexity of this system, [70] 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.

5.4 Line Detection

Many line detection algorithms stem from the road domain, to detect lanes. This is done for autonomous vehicles or safety systems for cars. Most algorithms are trained on datasets like TuSimple [72] or CULane [73], which include images taken in the front view of cars. While TuSimple is captured on US highways CULane's images are from Beijing. Figure 7 shows examples of TuSimple.

Figure 7: Example images of TuSimple with annotations [74]. Red lines are horizontal anchor lines. Green circles mark image coordinates, where road lanes and anchor lines intersect.

[75] proposed one of the first lane detection approaches on TuSimple. The proposed method detects lanes and classifies the kind of lane. Two cascaded CNNs are utilized, the first one is an encoder-decoder network for instance segmentation and the second one is a classification network. [76] uses an encoder-decoder network at the start, then an introduced horizontal reduction module, that transforms features into a row-wise representation. In the last stage, the network outputs two branches. One that predicts the horizontal location of lanes and another that outputs a confidence score. However, both of those approaches utilize encoder-decoder structures and are therefore similar to the techniques used for semantic segmentation. Other structures for lane detection include key point extraction, grid systems, or polyline regression. Research in those directions are described in the next three paragraphs, following projects with similar approaches for the rail domain.

Keypoint detection on the road

[77] proposed Line-CNN, which includes a ResNet backbone and an introduced "Line Proposal Unit". The model suggests a series of lines to locate road lanes. After the proposal, the line with the highest confidence score is fitted with horizontal offsets to match the actual lane. Similar to Line-CNN, the method introduced in [78] also proposes lines with confidence scores and fits the resulting output with horizontal offsets. The innovative aspect of this approach lies in the introduction of an anchor-based attention mechanism. This added module results in higher accuracy and a much faster model with speeds up to 250 FPS. [79] proposed a network, that predicts road lanes with the horizontal offsets from a pre-defined vertical anchor. Additionally, a Network Architecture Search (NAS) is utilized to find the optimal architecture for this task. Another approach that uses key points is introduced in [80]. This architecture utilizes a "Lane-aware Feature Aggregation Module" to strengthen the connection between neighboring key points and achieves state-of-the-art performance.

Grid system

An approach that uses grid systems is proposed by [81]. Here a row-wise classification is done for a predefined number of grids. The number of grids is equivalent to the number of lanes that can be detected.

Polyline regression

A regression approach is proposed in [82], that predicts the polynomials of lanes. ResNet and EfficientNet backbones are used to extract features, which are then used to forecast lines. Each line includes the coefficients of a polynomial, a starting parameter, and a confidence score. Additionally, a shared horizon line is predicted, where all polylines end. Compared to other state-of-the-art approaches, [82] proposes a computationally efficient method with speeds up to 115 FPS. While accuracy is lower than with other methods, it still is comparable. [83] proposed an approach that also predicts the parameters of a cubic curve. Instead of the polynomial coefficients, the parameters of Bezier curves are predicted. A Bezier curve is fitted in an area defined by four control points. For the network structure a typical backbone like ResNet with additional feature flip fusion modules is utilized. Then a pooling and two convolutional layers are implemented sequentially, which form the final output of predictions. Compared to polynomial fitting the approach proposed in [83] is superior.

Rail domain

Even though much has been done in the road domain, and the rail domain is usually not considered, some research has still been conducted to adapt lane detection algorithms for trains. Some of the mentioned research approaches utilize key point extraction to detect rails. [84] detects rails by adapting the concept of key point extraction with semantic segmentation as

pre-processing. The technique can be divided into four stages. In the first stage called "image preprocessing", a semantic segmentation network filters out the rails in an image. Additionally, an inverse perspective transformation is utilized to bring the binary mask into a bird-eye view for easier image handling in further steps. The second stage named "rail-track discretization", includes a key point extraction, along with connectivity judgment and breakpoint connection modules are used to divide the one-class segmentation mask into different rails. The third stage consists of the "rail lane reconstruction", where the data from the second step is combined with the bird-eye view from the first to connect the extracted key points. Resulting in a complete rail lane instead of discrete points. In the final stage, rails are matched in pairs, to receive the final output. However, no distinction is made between the train's rail and other rails. Two additional issues lay in the complex data processing of this approach. On the one hand, it is stated that the algorithm's accuracy relies too much on the semantic segmentation used at the start. On the other hand, no records of speed are available. Due to the intricacy of the proposed approach, the complex and cascaded algorithms could easily exceed real-time requirements.

Another state-of-the-art approach to filtering out rails is to divide the output into grids [85]. This technique is adapted from the road domain from [81]. Each rail is predicted in a separate grid and the number of rails is predefined with the hyperparameter C, as shown in Figure 8. In each grid, a row-wise classification problem is solved, where the target class corresponds to the cell that encompasses the target rail. After that, the predicted cells are translated into image coordinates to calculate the final output. The architecture is shown in Figure 8. The output image shows the focus on detecting all rails in an image. This approach relies only on the feature extractor and therefore does not need a decoder. In 2024 this approach is also used by TEP-Net [1] with only the train's own two rails ($C = 2$) for state-of-the-art comparison.

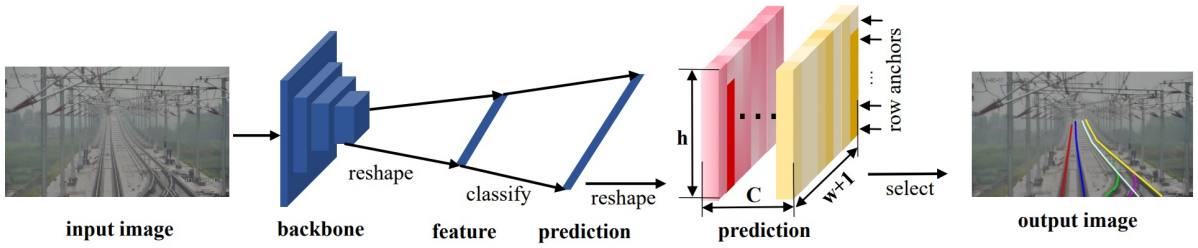


Figure 8: Rail detection with Gridsystem [85]. Hyperparameters: C is the number of rails (number of grids). h is the number of cells vertically. $w + 1$ is the number of cells horizontally with an additional column for a background class, when no rail is detected in a row.

TEP-Net [1] introduces another regression approach. [67] and [70] expect that switch states cannot accurately be determined. Therefore they try to filter out all possible tracks. [1] extends the problem formulation with the assumption that switch states can be correctly predicted. This eliminates the need for heavy post processing and leads to a lightweight model that only predicts the right and left rail of the trains path. Since the use case of [1] and the goal of this work overlap in almost all aspects, this paper is described in more detail in section 5.5.

5.5 Baseline Paper

To do justice to the contributions of [1], 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 only filters out the two rails the train continuous on, presenting a solution to the rail track prediction problem. Contrary to most literature, in which often all rails are detected without any distinction between different rails. Efforts in rail detecting with the trains path in consideration have been made in [67] and [70]. However, because of the assumption that switch states cannot be accurately determined all "possible rail tracks" are considered in those two papers, leading to complex post-processing. In [1], 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. Contributions include: new annotations tailored for this use case, two data augmentation strategies, a novel model architecture and a custom loss function.

New annotations are labeled for this project, which only cosiders the "ego-path". Images are taken from the RailSem19 dataset [1]. For further details of the dataset, please refer to section 5.8.

Two data augmentation strategies are deployed: one for training and one for inference [1]. In the training along with usual methods like image color variations and horizontal flips, a cropping mechanism is implemented, which is dependent on the GT. This allows to only focus on the most relevant part of the image. Since, only crops are used for training, the use case in inference must reflect something similar. Therefore an autocrop method is developed, which crops images according to a running average of previous predictions. Both of those techniques are described in section 6.4 in more detail.

The model architecture introduced by [1], includes a backbone for feature extraction followed by a prediction head. The head is constructed out of fully connected layers and forms the output at the end. The output vector, includes the x-values for the left and the right rail on anchor lines and a value for the horizon line. Figure 9 shows the proposed regression model in the middle. A more detailed describtion is given in subsection 6.1.2.

The loss function is tailored to the regression approach and is constructed of two functions. The trajectory loss, which is responsible for the horizonal error and the y-limit loss, which takes the horizon line into account. Both losses are weighted and addded. For a detailed describtion, plese refer to section 6.5.

5.5.1 Comparison other state-of-the-art approaches

[1] compared its appraoch to promising and common methods in the literature: a segmentation and a classifiaction approach. To fairly compare the novel regression-based architecture, [1] uses the same backbone and replaces the prediction heads, as illustrated in Figure 9. The segmentation model utilizes a U-Net-like [57] decoder, 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 [85] and follows their settings. It outputs a $2 \times 64 \times (128 + 1) = 16512$-dimensional vector. 2 grids for 2 rails, with a height of 64 and a width of $128+1$. The $+1$ is for the background class. A cross-entropy loss is used to train this model [1]. For experiments, the same dataset is utilized with various preprocessing steps to fit the specific task of the model.

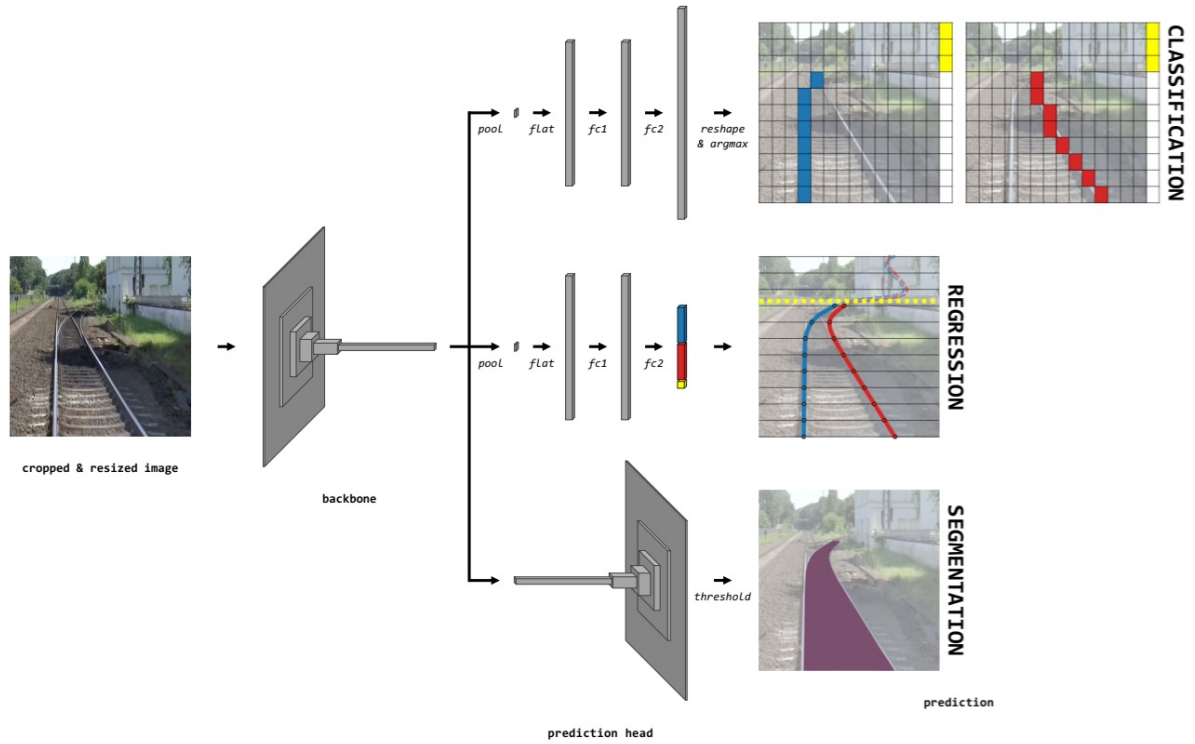


Figure 9: The model architecture is designed to enable a fair comparison between the novel regression model proposed in [1] and other State Of The Art (SOTA) approaches. All models use the same dataset with preprocessing steps to fit annotation to the model task.

Experiments include trainings with different versions ResNet and EfficientNet backbones. According to the Intersection Over Union (IoU) the segmentation model is the most accurate [1]. However, the difference in performances is within a range of only 1.4 percent. Classification with ResNet18 performs worst. While segmenation with EfficientNet-B3 achieves the highest accuracy, it is also the slowest. In terms of speed the regression-based approach outperforms other models. Additionally, it proves to be light weight because of lower number in parameters and Multiply-Accumulate Operations (MACs). These characteristics are of great importance for the rail track prediction application of this work. For more detailed results, please refer to [1].

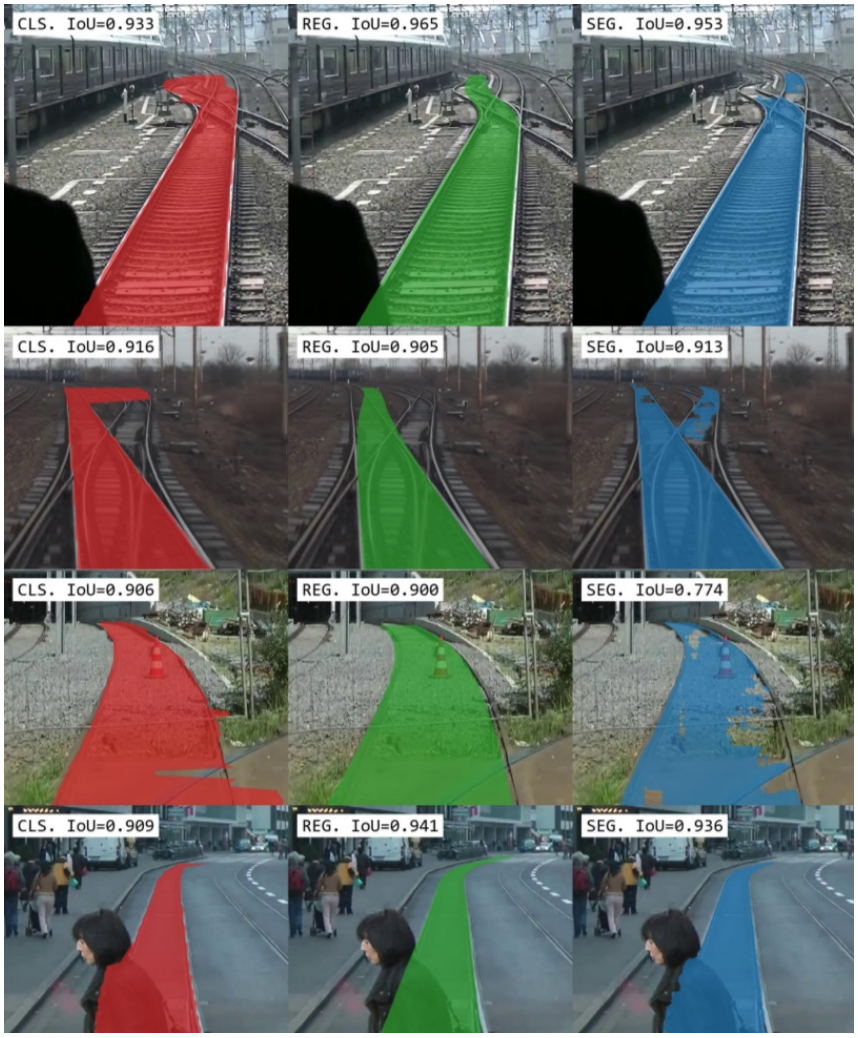


Figure 10: Comparison between classification (CLS), regression (REG) and segmentation (SEG) models with challenging scenes. The worst backbone (ResNet18) is used for this figure to clearly show difference in behaviours [1].

Since all three model architectures achieve similar IoU scores, [1] compares performances on individual challenging scenes. Figure 10 visualizes a drop in accuracy when the model becomes unsure. However, the regression model is the only one which keeps the form of a rail when the track splits and seems to no issues with obstructions. This is because 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. The cross-entropy loss for classification and the dice loss for segmentation both penalize misclassifications but do no account for increasing distance from the GT. These models work with probabilities, which tend towards extrems under certainty. When uncertain segmenation models move closer to a threshold and classification models show more spread-out probabilities across classes. The regression appraoch inherently involves continuous values, which assumes averages among uncertain possibilities. Resulting in a more robust system [1].

5.5.2 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 11. 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.

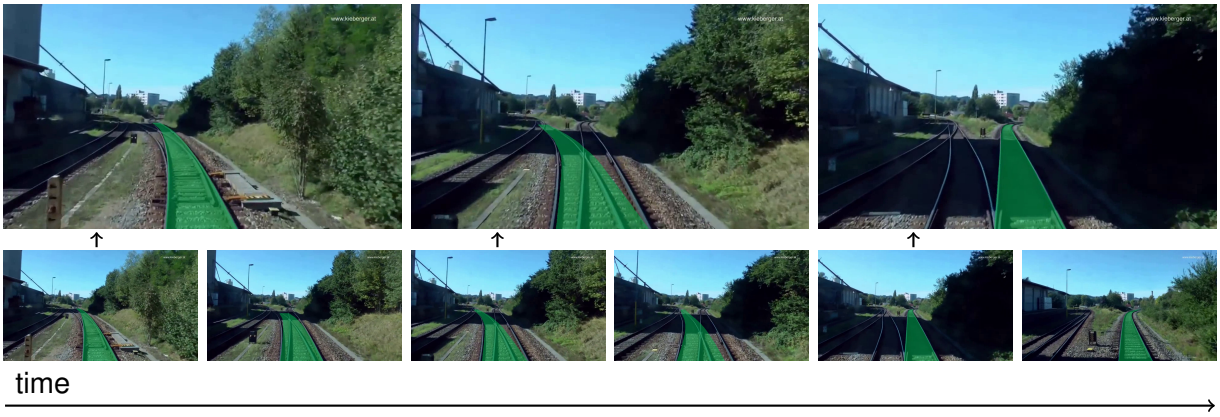


Figure 11: Limitation of TEP-Net [1]. The introduced approach is a single-frame-based model. Therefore, no temporal context can be captured, which leads to uncertainty in prediction when driving over a switch. All images are from [86]. A YouTube video, which is also used by RailSem19. It is ensured that 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 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.

5.6 Network architectures

One of the first CNN architectures was the **Le-Net5** [87]. 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) [88] 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 [89].

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)** [55] used small three-by-three filters and many layers providing a deep network architecture. Secondly, the **Inception** network or **GoogLeNet** [90]. To extract features in a range of various scales, this architecture utilized different filter sizes.
- In 2015 the **ResNet** architecture [51] 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 [89].
- In 2016 **DenseNets** [71] 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 [89].
- In 2017 **MobileNets** [91] [92] [93] 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 [89].
- In 2019 **EfficientNets** [94] was introduced to further explore greater trade-offs between accuracy and efficiency, by using scaling methods for model architectures [89].

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.

5.6.1 ResNets

A widely used model architecture in the community is ResNet [51]. 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 12. 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 [95]. However, ResNet models are still resource-intensive because of their size [89].

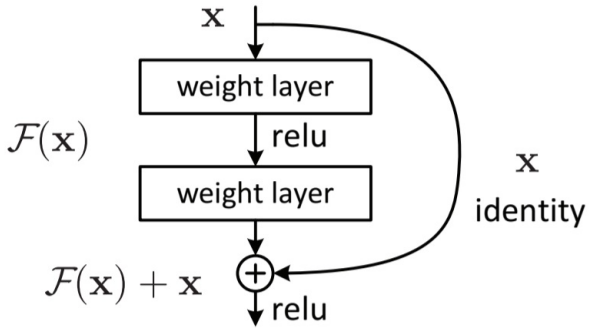


Figure 12: ResNet residual block [51]

Figure 12 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.

5.6.2 DenseNets

DenseNet [71] 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 13. 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 [96].

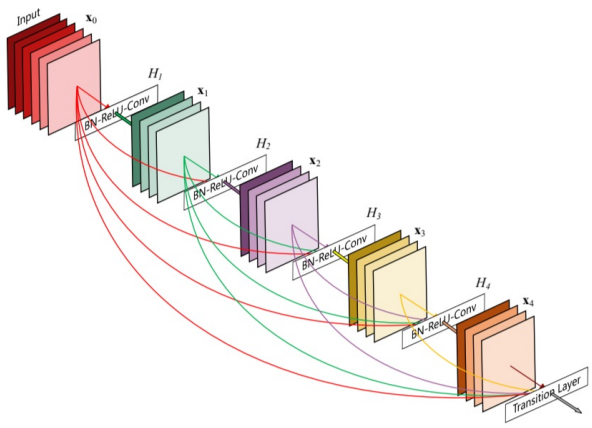


Figure 13: DenseNet dense block [71]

5.6.3 MobileNetV3

The goal of the MobileNet series [93] 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 [89], 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 [89]. 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 14a.

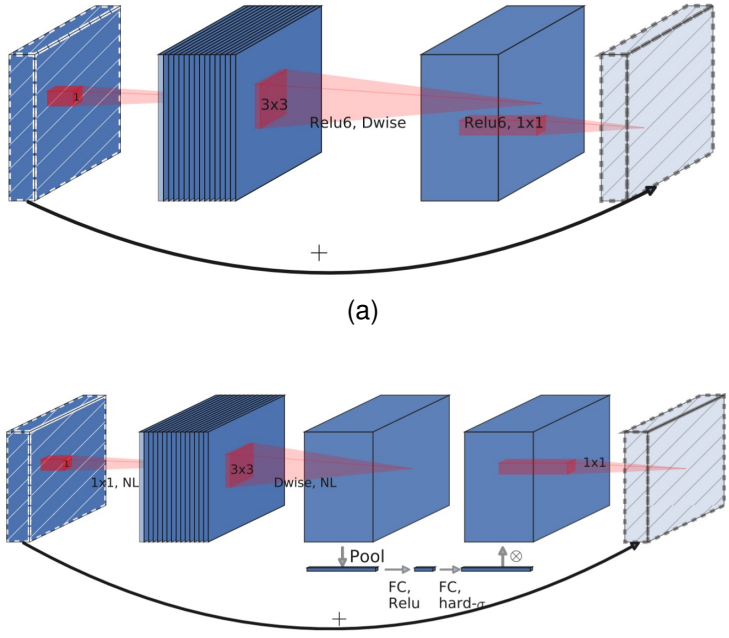


Figure 14: MobileNet V2 and V3 Blocks [93]: **(a)** MobileNetV2 bottleneck block, **(b)** MobileNetV3 bottleneck block with added Squeeze-and-Excite block

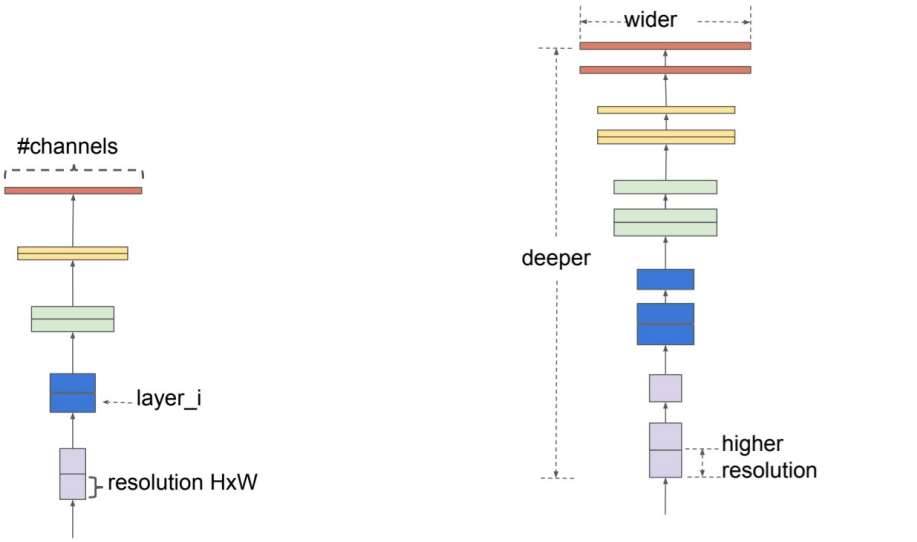
MobileNet V3 added optional blocks called Squeeze-and-Excite, which is illustrated in Figure 14b. These blocks can be added to any CNN and consist of a global pooling layer and two fully connected layers with a ReLu and a Sigmoid activation function respectively. After that, the output of this additional block is multiplied with the input feature map of the SE block. This enables the weighing of specific channels [69]. Additionally, for the MobileNet V3 a Network Search is used to find optimal network structures and experiments with various activation functions have been conducted [93].

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 [97].

5.6.4 EfficientNets

EfficientNet [94] further investigates methods to find CNN architectures optimized for efficiency. Inspired by MobileNet V3 the models also deploy residual bottleneck convolutions. [94] calls such a block an "MBConv". Since [94] observed the correlation between performance improvement and model scaling, the idea is to strategically adjust the depth, width, and resolution of a network. Figure 15b 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 [98].



(a) network before scaling method

(b) scaling parameters of a network acording to [94]

Figure 15: Systematic visualisation of scaling parameters of a network [94]

5.7 Temporal Model Architectures

- LSTM -> 1/2 Seite
- GRU -> 1/2 Seite
- Approaches 1/2 bis 1 Seite

2020 Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks

- Encoder
- LSTM
- Decoder

2021 Lane Detection Model Based on Spatio-Temporal Network With Double Convolutional Gated Recurrent Units

- Encoder
- GRU
- Decoder

2022 A hybrid spatial–temporal deep learning architecture for lane detection (U-net based)

- Encoder
- ST-RNN blocks
- Decoder

2022 Sequential Neural Network Model with Spatial-Temporal Attention Mechanism for Robust Lane Detection Using Multi Continuous Image Frames

- Encoder
- attention module:
  - skip connections
  - fc layer
  - LSTM
- Decoder

2023 Robust Lane Detection through Self Pre-training with Masked Sequential Autoencoders and Fine-tuning with Customized PolyLoss (https://paperswithcode.com/paper/robust-lane-detection-through-self-pre)

- self pre-training
  - pretraining for lstm initialization
- fine tuning with segmentation architecture:
  - Encoder
  - LSTM
  - Decoder
- post-processing:
  - DBSCAN
  - Curve Fitting

5.8 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 [99] consists of small images with pixel size 32x32. This dataset includes 600 images with the label *trains*. In *PASCAL VOC2012* [100] there are 544 images labeled *trains*. *Microsoft COCO* [101] 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* [102] includes labels like *electric_locomotive* (4330 images), *steam_locomotive* (1187 images) and *bullet_train* (1205 images). *Open Images Dataset V4* [103] 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 [104] 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* [105] 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* [106] includes the same 182 classes like the *Microsoft COCO* [101] 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* [107] has the same dense labels like *Cityscapes* [104]. 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 [108].

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 [109] [110] or even some to detect garbage in grooved rails [111]. Then there are datasets in ego-perspective

of the train driver like *FRSign* [112] or *GERALD* [113], 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 16: 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 [60]. 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 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 17: Example images and GT of RailVID dataset [60]

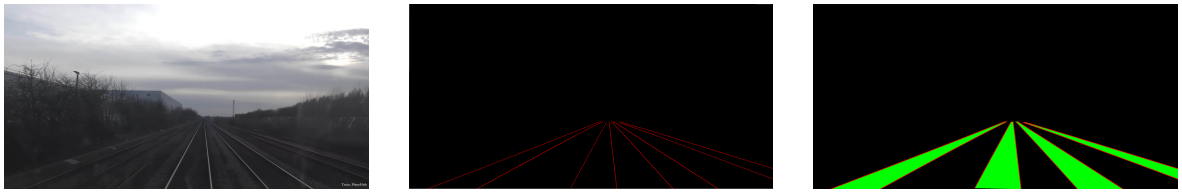
RailSet

[59] and [58] presented the *RailSet* dataset, which is divided into two sub sets: RailSet-Seg for segmentation and RailSet-Ano for anomaly detection [59]. Both of them are captured in the ego perspective of the train driver [59] [58]. 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 [59]. 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 [59]. 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 [59] [58].

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 [58]. 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 18: RailSet-Seg example with annotations [59] [58]: **(a)** raw-image, **(b)** rail class, **(c)** rail and rail-track class

RSDS

The creators of the 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. 19 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 19: RSDS example with annotation [27]: **(a)** raw-image, **(b)** ground truth

Rail-DB

A very similar dataset is Rail-DB [85]. 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. 20 shows example images of Rail-DB's different scenes. However, it seems that the images again have very specific characteristics like in RSDS. 20 also shows, that all images are captured on a Chinese line. Even though [85] 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 [85] states in the GitHub repository [114], that this project fails to generalize on example videos, where the scene looks different from the one the dataset is captured on.

Figure 20: Rail-DB [85] images with annotations in different conditions

OSDaR23

Another dataset it the OSDaR23 [115]. 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 [115]. 21 shows the 11 different frames of each sensor and 22 shows an example of an annotated scene. As illustrated in 22 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.

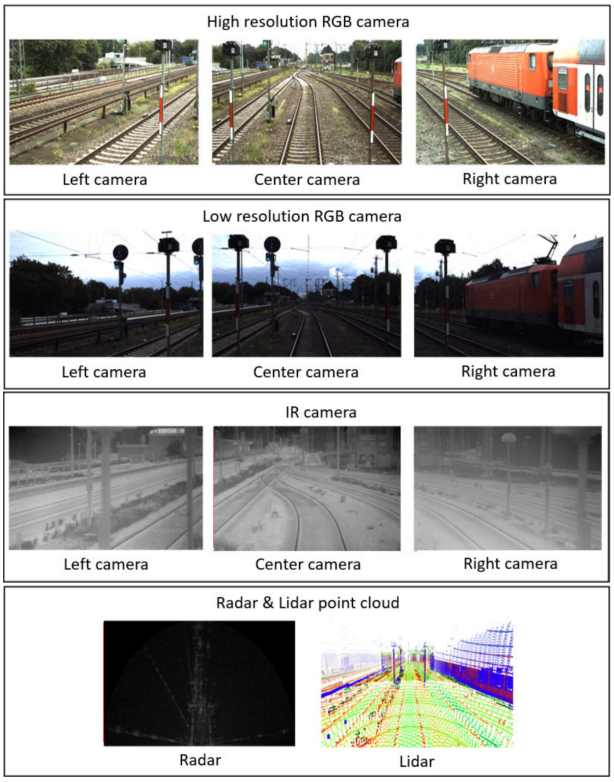
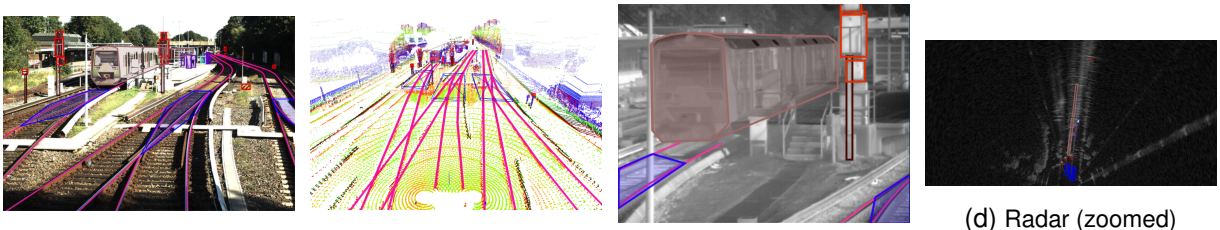


Figure 21: OSDaR23 data from all different sensors [115]



(a) RGB center camera (b) merged lidar point cloud (c) IR center camera (d) Radar (zoomed)

Figure 22: OSDaR23 annotated scene [115]

TEP-Net dataset

[1] presents the 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 23. 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 always 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 24. 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 23: Examples images from RailSem19, which are not included in the TEP-Net dataset due to unclear circumstances about the trains direction. [1]

Figure 24: TEP-Net dataset example images with annotation [1]

6 Methodology

In this section, the methodology of this work is described. In section 6.1 used and implemented models are explained. section 6.3 describes datasets, which are used to train and evaluate models and other algorithms. The data augmentation of these datasets is depicted in section 6.4. section 6.5 deals with loss functions. After that, setups for training and inference measurements are described in section 6.6 and section 6.7. Keep in mind that the main contributions of this work include three parts:

1. An initial approach to combine object detection with semantic segmentation.
2. TEP-Net [1] is the new baseline for further improvements in accuracy and robustness.
3. An RNN is integrated aiming to eliminate the limitation of single-frame-based approaches.

6.1 Models

In this section all models are discussed which are used or created in this work. This section is divided into three subsections. Each of them corresponds to one of the main contributions of this work. Therefore the first section gives a brief overview of used object detection models, the second one with the TEP-Net [1] because it is used as a baseline. Additionally, further improvements of this model are described in this section. The last section describes all temporal models.

6.1.1 Object Detection Models

The state of the art in section 5.2 shows that one-stage detectors have the most promising characteristics to be successfully utilized in this work. The most interesting ones are the models from the YOLO series, especially the most recent ones because they focus on improved parameter usage. This makes the models light weight and presents an advantage for this work, because operation on limited hardware is aimed for. At the start of this work the most recent model from the YOLO series is the YOLOv9 [45]. For object detection different models of the YOLOv9 and the GELAN series are available. However, the YOLOv9 models were not fully supported at that time. Therefore the models used for experiments include the GELAN-c and GELAN-e. These are obtained trough the GitHub repository [116] and used unchanged. An additional model which is used for experiments is the YOLOv7 [44] model, which is also utilized unchanged from the GitHub repository [117].

6.1.2 TEP-Net Model

In the literature rails are often detected without distinction between all rails visible in an image and the rail the train continues. [1] therefore proposes 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 to the rail domain. Since, this method fits well to the goal of this work, it is used as a baseline for further experiments.

Although rails can be represented by second or third-degree polynomials most of the time [82], 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.

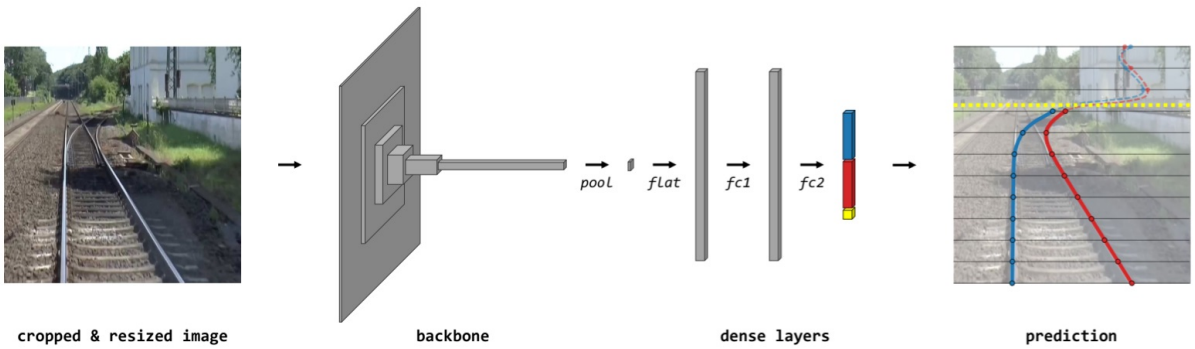


Figure 25: TEP-Net model architecture[1]. The input of the model is a cropped and resized image and the output of the model are the x-values for the left and right rail on each anchor line plus an additional value for the y-limit.

As shown in the prediction of Figure 25, 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 prediction image of Figure 25 and the second and fourth images in the bottom row of Figure 37 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. The input of the backbones is an image crop with the size of $3 \times 512 \times 512$ which represents a rough average of crops obtained by data augmentation strategies. These backbones extract relevant features from these crops 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 [118] and flattened to a vector. This feature vector serves as the input for the prediction head, which consists of two fully connected layers [119] 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 $2 \times H + 1$. This vector includes the entire information of one prediction. H is the number of anchors. 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.

6.1.3 Improvements to the TEP-Net Model

One of the main contributions of this work is to improve the chosen baseline model from [1]. These improvements can be roughly divided into three parts. First, the backbones are exchanged with other architectures. Secondly, after the backbones, a true pooling layer is integrated. Thirdly, experiments with different prediction heads are conducted. The following sections describe the structure of these architecture changes in detail.

Backbones

Backbones are used to extract features from images. These are parts of CNNs that input and output tensors. Common CNN architectures usually work with 4D tensors with $B \times C \times H \times W$ as shown in Figure 26. B is the batch size, C is the channel size, H and W are the height and width of the input image. The backbone transforms this high-level feature map into a low-level one with different operations, resulting in a tensor with a high number of channels C and low numbers in resolution H and W. The batch size B tells how many images the CNN processes simultaneously and stays the same. These characteristics apply to all four chosen backbones. As described in the state of the art in section 5.6, the most interesting models are ResNet [51], EfficientNet [94], DenseNet [71], and MobileNetV3 [93]. ResNet and EfficientNet are already implemented in [1]. DenseNet and MobileNet are additionally integrated in this work. Table 5 gives an overview of all different versions which are implemented in this work.

ResNet	18, 34, 50
MobileNetV3	small, large
DenseNet	121, 161, 169, 201
EfficientNet	B0, B1, B2, B3

Table 5: Versions of backbones utilized in this work

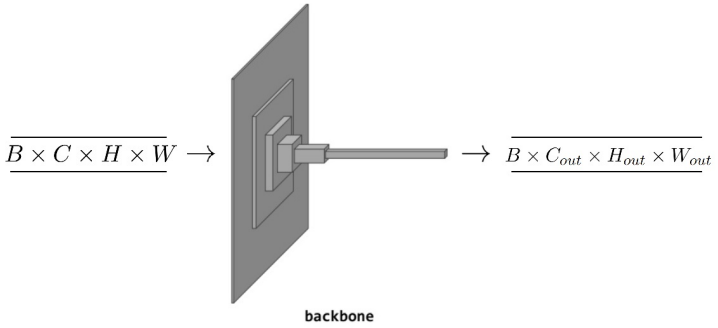


Figure 26: Backbone logic (Backbone graphic taken from [1]).

When exchanging backbone architectures it is important, that the dimensions of input and output tensors are the same for every backbone otherwise the next layers would get different input dimensions than they are designed for. Standardizing the input tensor is done by resizing an image to 512×512 after augmentation. While B is set by a hyperparameter, $C = 3$ because images are in the 3-channel RGB format. For the output dimensions, the backbone architectures and their operations must be analyzed. When inspecting all four backbones in more detail it becomes clear that they all share the following characteristics:

- The input resolution in all papers is 224×224.
- At the end, there is either a pooling layer, a 1×1 convolutional layer, or both in combination with fully connected layers forming a prediction head for classification tasks.
- Before the prediction head, the spatial resolution of feature maps is 7×7.

With this information, the last layers of each backbone can be discarded, so all backbones output tensors with the same H and W. In all four papers, the architecture transforms tensors with a resolution of 224×224 to a tensor with 7×7 before the prediction heads. This means they all have the same reduction factor of 32. Since the input resolution of 512×512 is used in this work resulting feature maps have a resolution of 16×16 after backbones. The exact dimensions of all tensors are shown in Table 6.

Backbone	**Input dimensions**	**Output dimensions**
ResNet	[8, 3, 512, 512]	[8, 512, 16, 16]
MobileNetV3		[8, 96, 16, 16]
DenseNet		[8, 1024, 16, 16]
EfficientNet		[8, 384, 16, 16]

Table 6: Versions of backbones utilized in this work

Pooling Layer

In section 6.1.3 the spatial resolution is standardized across multiple backbones. However, Table 6 shows that after different backbones different numbers of channels C are obtained. To solve this and make the backbones interchangeable, [1] proposed a pooling layer at the start of the prediction head. However, when analyzing the code [120], it becomes clear that [1] used a convolution with a 1×1 kernel instead of a pooling. This *conv2D1x1* transforms the feature map of the backbone to a predefined channel size. In [120] it is set to 8, resulting in the dimensions $8 \times 16 \times 16$. Then this feature map is flattened to a vector for further processing with fully connected layers.

A critical positional encoding is missed when directly flattening this feature map. Therefore in this work after standardizing the dimensions with the *conv2D1x1* layer, a true pooling layer is integrated. This additional layer consists of either an adaptive average pooling [121] or an adaptive max pooling [122]. An additional encoding takes place resulting in a vector-like tensor with dimensions $C \times 1 \times 1$. Besides the extra positional encoding, a further gained advantage is that the structure of data stays the same when flattening this feature map to a vector.

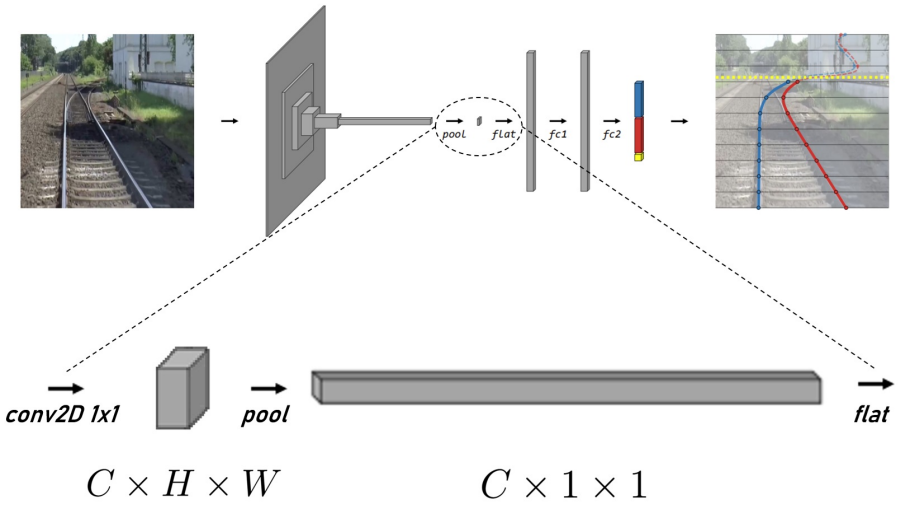
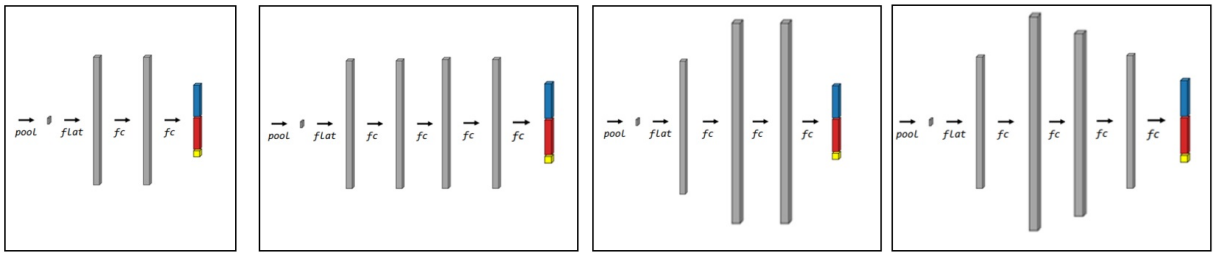


Figure 27: Integration of a true Pooling Layer (Model graphic taken from [1] and changed).

prediction Heads

Another improvement to TEP-Net [1] is a different prediction head. Experiments are conducted with four variations of architectures shown in Figure 28.



(a) Original Head [1] (b) Depth Head (c) Width Head (d) Trapeze Head

Figure 28: Different architectures for prediction heads.

For comparison, the original head is displayed in Figure 28a, which has 2 Fully Connected (FC) layers [119]. The first one has input and output sizes of 2048 and the second one reduces the 2048 features to the output size $2 \times H + 1$ as explained in subsection 6.1.2. The Depth and Width head, shown in Figure 28b and Figure 28c are inspired by the EfficientNet [94] scaling technique. The Depth head increases the number of FC layers to 4 and keeps the feature size at 2048. The Width head doubles the feature size of layers. Before that, a FC layer gets the flattened feature vector from the backbone of site 2048 and increases it to 4096. Then two wider FC layers are used, while the second one reduces the features to the output size. Figure 28d shows the Trapeze head. This had is named after its rough form and is a combination of the Depth and Width head. First, it increases the number of features then it gradually compresses data until the last FC layer reduces the size to the defined output. The exact input and output size of each layer is shown in Table 7. All heads use linear layers [119] and Rectified Linear Unit (ReLU) activation functions [123] in between visualized in Figure 29.

Layers	**Input size**	**Output size**
1. FC-Layer	2048	3584 (2048×1.75)
2. FC-Layer	3584	2560 (2048×1.25)
3. FC-Layer	2560	2048
4. FC-Layer	2048	$2 \times H + 1$

Table 7: Detailed network structure of Trapeze Head

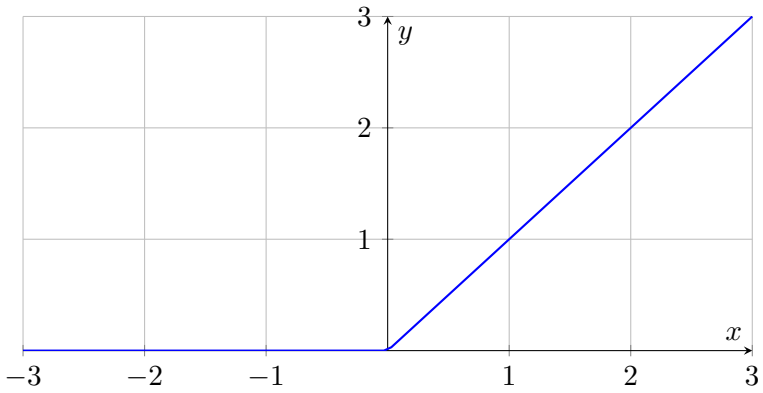


Figure 29: $ReLU$ Activation Function [123]

6.1.4 Temporal Models

As described in subsection 5.5.2 there is a limitation to this model presented by the single-frame-based approach. This becomes a disadvantage especially when focusing on switches. While there are many possible approaches to dampen this issue, the most promising solution incorporates an RNN in the model. The RNN module should consider the temporal component and learn the dependencies between successive frames. Therefore, this CNN-RNN architecture inputs a video stream or in simpler terms, several consecutive frames instead of a single one. Since the wanted prediction output is the same as the single-frame-based model, it stays the same. This effectively makes this approach a sequence-to-one problem, in which only the output of the last frame is considered. A schematic visualization is shown in Figure 30 in which the RNN module is in the prediction head before the FC layers.

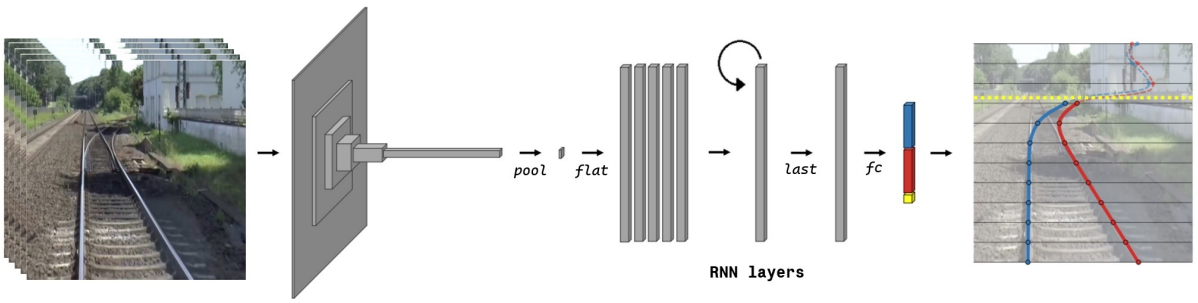
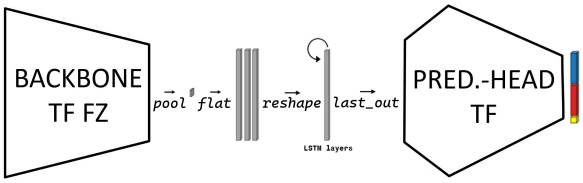


Figure 30: Schematic drawing of a temporal model with an integrated RNN module (Graphic taken from [1] and changed).

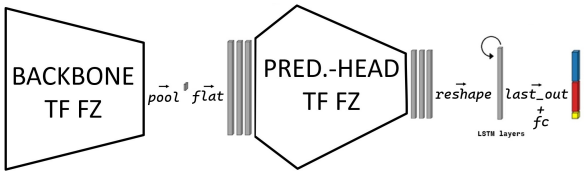
The new problem formulation demands the development of new implementations and solutions. However, the single-frame-based model shows satisfactory performance in almost every scenario except the ones including switches. This led to the concept of reusing all parts of the single-frame-based model training, as long as the new temporal problem formulation allows for their re-usability. Since the new formulation deals with a sequence-to-one problem, the output is on a single frame. This allows the loss function to be used as it is. Another idea

is to not waste any past experiments or model trainings. Therefore, all temporal models use as many pre-trained weights as possible resulting in transfer learning approaches up to the point where in the model the temporal component is learned. This means the weights from the best single-frame-based model are used for initialization. Furthermore, in all experiments, backbones are frozen. This strategy does not allow weights to be changed during training. The idea is, that backbones are already trained to extract the most relevant features of images from single-frame-based trainings.

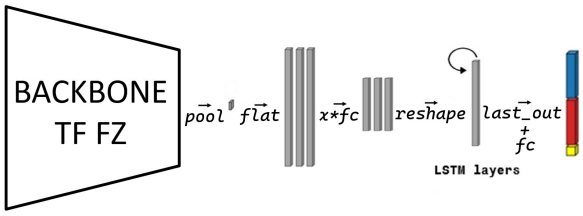
Following these rules leads to 8 different model structures, which have been trained and evaluated on a new dataset tailored for this use case. There are two kinds of temporal models tested for this application. The first kind incorporates RNN modules. For the second kind, the idea is to learn temporal dependencies only with FC layers. All models are visualized in Figure 31.



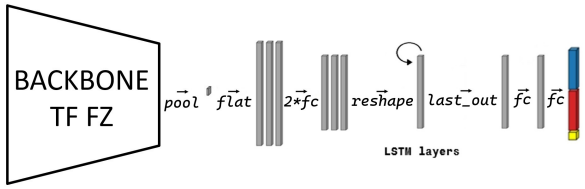
(a) CNN_LSTM_FC



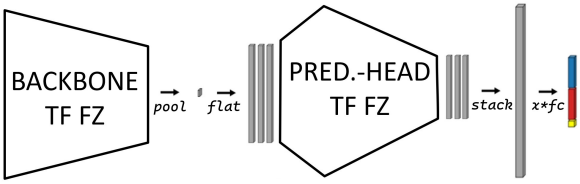
(b) CNN_FC_LSTM



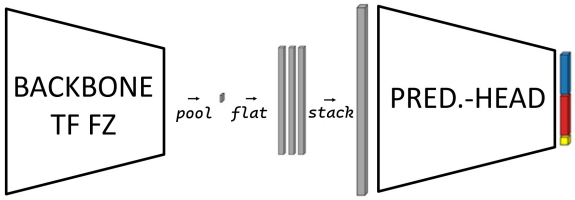
(c) CNN_LSTM: $x = 1$ (V1), $x = 3$ (V2)



(d) CNN_LSTM_HEAD



(e) CNN_FC_FCOUT: $x = 1$ (V1), $x = 3$ (V2)



(f) CNN_FLAT_FC

Figure 31: 8 different temporal models for experiments. **(c)** and **(e)** have two versions in which the number of FC layers vary. One FC layer is used in for V1 and three Layers are used for the V2 model. Models **(a)**, **(b)**, **(c)** and **(d)** include LSTMs modules. Models **(e)** and **(f)** try to learn temporal dependencies only with FC layers. Shape of heads resemble the architecture e.g. trapeze shaped contour is the trapeze head shown in Figure 28d

Since the temporal modules need data in other formats, the logic of data processing before and within the model has to be changed as well. The comparison between the data handling

of single-frame training and the one proposed in this work for temporal training described in section 6.2. In typical models that take single images into account, the network gets tensors with four dimensions consisting of batch size B, channels C, high H, and width W also shown in Figure 26. However, in this work, an additional dimension of the video stream length is introduced. S therefore defines how many sequential frames the model is fed with, resulting in a 5D tensor $[B, S, C, H, W]$. Since the backbone is the same as the single-frame-based model it only supports 4D tensors shown in Figure 26. For that reason, before the backbone, the tensor is transformed into a 4D one with the view function [124]. B and S are multiplied, resulting in $[B \times S, C, H, W]$. This feature map is then processed by the backbone. After flattening the tensor, only two dimensions remain $[B \times S, C]$. In Figure 31 this feature map is visualized with three blocks indicating multiple feature maps from numerous images. It stays that way until either a $reshape$ or a $stack$ transformation is conducted. In this work, $reshaping$ is done by transforming $[B \times S, C]$ to $[B, S, C]$ as this form is needed for LSTMs. $Stacking$, on the other hand, stacks the feature map into one big vector, resulting in a tensor where all images are in the C dimension.

LSTMs output sequence tensors with the same dimensions $[B, S, C]$. Since only the features of the last sequence part are considered, the $last\_out$ transformation takes the channels of those images resulting in $[B, C]$ which is then further processed by FC layers. Even though almost all transformations support multi-batch training, to ease the logic all experiments with temporal models are done with a batch size of 1. Similarly, when inferring these models a video stream consisting of multiple frames is transformed into a 4D tensor with dimensions $[S, C, H, W]$. Therefore, to support inference this tensor must be extended with a batch size of 1 resulting in a tensor with dimensions $[1, S, C, H, W]$, which the model can take as input.

6.2 Data handling and Training process

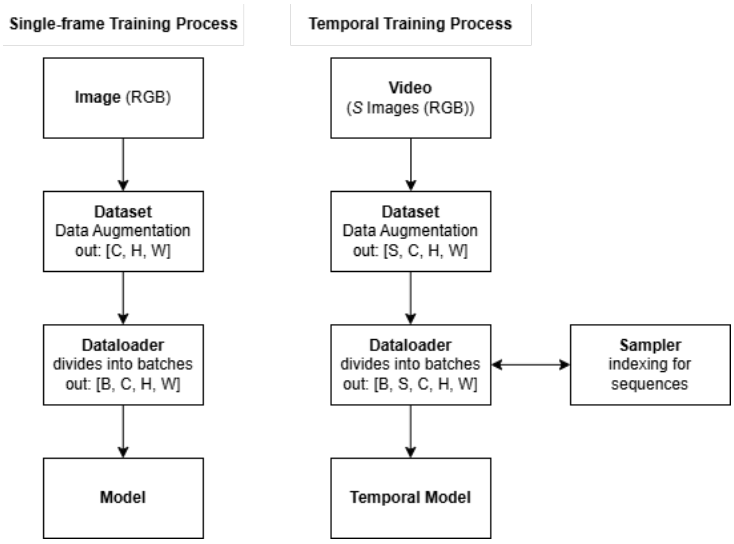


Figure 32: Comparison between data handling processes for single-frame training and the proposed temporal training.

The Dataset class reads in images according to an index, performs online data augmentation and outputs a tensor. In the Dataset class also the targets are formed out of the augmented annotations. First Polylines are transformed into a mask, then the data augmentation from section 6.4 is applied. In the last step this augmented mask is used to generate regression or classification targets. For semantic segmentation the mask can be utilized directly. The Dataloader iterates through the dataset step by step and calls the dataset class using indices. It can either move step by step or rely on a sampler to dictate the stepping pattern. In temporal training, a sampler is essential for managing sequences, avoiding out-of-bound errors and preventing scene intermixing. A sliding window approach is implemented, shown in Figure 33. A window of a predefined number of images serves as input for the model. This number (usually 10) is smaller than the sequence length 76. Then the data loader steps through the sequence. At the end of a sequence the sampler forces the Dataloader to jump to the next sequence instead of mixing them. Furthermore the Datalaoder creates batches and therefore adds the dimension B to the tensor.

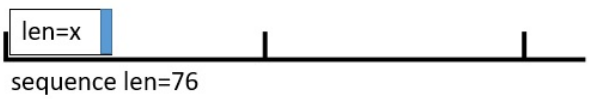


Figure 33: Sampler is responsible for the sliding window approach in temporal training: bottom lines represents the sequence with a length of 76 frames; the box is the window of frames used as the input for the model with x being the length of this window (with $x \leq 76$); the blue part of the sliding window represents the output frame, because this temporal method deals with a sequence-to-one problematic.

The training consists of a pre-defined number of epochs, which presents one training cycle. In each epoch, the Dataloader is used to feed the model one batch. Then the model predicts, calculates the loss according to this prediction and performs the back propagation to optimize the model. In the same cycle the validation is done, in which the Dataloader feeds the model samples of the validation set. Here only the prediction is done and the loss is calculated. Two additional steps are done in an epoch after training and validating. On the one hand, the scheduler updates the learning rate. In this work the OneCycleLR scheduler [125] is used, which calculates the used learning rate according to a formula. On the other hand, when the validation loss reaches a new minimum, the model parameters are saved. This way the model with the lowest loss is obtained at the end of all epochs.

6.3 Datasets

In this section all datasets are described that are used in this work. The first subsection describes which subsets of RailSem19 are used for the first attempt for the training of object detection models. Then the annotations of [1] are briefly discussed, which are used to train all single-frame-based models. After that a switch evaluation dataset is described that is used to evaluate singe-frame models on switch scenarios. The last subsection deals with the temporal dataset and its creation.

6.3.1 RailSem19 and subsets for training object detection models

The first approach to creating a Rail Track Prediction system includes combining an object detection model with a semantic segmentation model. Since RailSem19 supports both of these methods this dataset is chosen for training. Furthermore, state-of-the-art research in section 5.8 shows that this dataset is the only one that not only includes switch annotations but also labels for switch states: $switch\_left$, $switch\_right$. Since not all states are identifiable even though a switch is visible there also is the $switch\_unknwon$ label.

In this work, the bounding boxes are utilized to train the YOLO object detection models. Experiments have been conducted with three versions of this dataset. An overview of these subsets is shown in Table 8. The first set is the whole dataset, which includes 8500 images with all different bounding box labels. The second subset only considers images with switch labels including the $switch\_unknown$ label. For this subset, 2764 images remain. The third subset only considers the switch labels in which the state is identifiable. This is because when focusing on the train's direction, the $switch\_unknown$ label does not hold any valuable information. To prevent confusion, all images containing $switch\_unknown$ annotations are intentionally excluded, resulting in a final set of 1,240 images.

RailSem19	**RailSem19_onlySwitches**	**RailSem19_onlySwitchesLeftRight**
8500 images	2764 images	1240 images
all bounding box labels	$switch\_left$	$switch\_left$
	$switch\_right$	$switch\_right$
	$switch\_unknown$	images with $switch\_unknown$ excluded

Table 8: Used dataset subsets of RailSem19 for training YOLO object detection models

Since the experiments of training object detection models showed unsatisfactory results also described in <section>, this methodology was deemed ineffective. This leads to the pursuit of a different solution and experiments with semantic segmentation models are therefore not accomplished. Consequently the corresponding dense labels of the RailSem19 dataset are not utilized.

6.3.2 TEP annotations

For training single-frame-based models [1] published its annotations for the images of RailSem19. These annotations consist of polylines for the left and the right rail of the train. Only the two rails are included in the labels which the train drives on. This is also the case when switches are present. Additionally, some images are excluded from this dataset when the train's track is not identifiable. For labeling this dataset the online tool CVAT [126] is used. These annotations can then be transformed with pre processing step to support other models like semantic segmentation. This dataset is described in more detail in section 5.8.

6.3.3 Switch evaluation dataset

For a practical application, it is important to make the output as useful and visible as possible. Therefore, the output should be the whole track, not only the rails. This is realized in the post-processing by filling out the area between the rails resulting in a binary mask. This mask is similar to the output of semantic segmentation techniques. It gives each pixel either the class label $track$ or no_track. For these methods, the most common evaluation metric is the IoU, which is also used by [1] to evaluate models on the test set.

This metric has its justification for general rail track prediction scenarios because it is a good indicator of model performance. However, when switches are present in the scene the IoU often loses meaningfulness. This issue is visualized in Figure 34. In this example, the model cannot correctly predict the train's path at the switch. Contrary to this uncertainty, the IoU still gives a high value indicating a good performance even though the direction of the train is wrong. The reason for this lies in the calculation of the IoU and this specific problem case. Since the switch is in the distance and the track is mostly correct up to this point, the predicted mask and the

mask of the GT still share a great portion of their areas. This leads to high values of the IoU metric, even though the track is incorrect.

Figure 34: Example of a switch scene that is incorrectly predicted, but the IoU metric still returns high values presenting potentially misleading performance scores. Track continuous to the left.

Since this work especially focuses on correctly predicting the train's direction in scenarios with switches, a solution to this evaluation issue must be found. Therefore a switch evaluation dataset is created, which solely focuses on switch scenarios. Furthermore, a points system is created that gives insight into the model performance. This point system is shown in Figure 35. The dataset consists of all images that contain switch cases out of the validation and test dataset that are obtained from the $80\% - 10\% - 10\%$ split of [1]. This results in 67 images which include 72 switches in total. The model gets 1 point for each correctly predicted switch. There are cases where models correctly filter out the track and set the horizon line before the switch. This behavior is rewarded with 0,5 points. This behavior is assumed to come from the labeling policy with $switch\_unknown$ labels. In [1] in images with $switch\_unknown$ labels, the track is annotated up to this label. This shows that the model is uncertain about the switch state. However, setting the horizon line before the switch is preferred over predicting the track incorrectly. In other cases, the model does not get any points.

(a) 0.5 points

(b) 1 point

(c) 2 points (2 switches)

(d) 0 points

Figure 35: Scoring system of the switch evaluation dataset.

This switch dataset deals with a qualitative performance evaluation and no technique is found to automate this process. Therefore, a model predicts the track of each image, and the points are given by manually observing outputs. Since the prediction process should resemble a real application, the autocrop technique is utilized. By predicting each image 50 times crop coordinates have time to converge to a crop similar to one in real applications.

6.3.4 Temporal Dataset

To train temporal models a dataset must be used that consists of videos and not only single images. [1] stated that there is no public temporal dataset available for the rail domain, which can be used for this use case. This statement is correct, to the best of the authors knowledge. Therefore a YouTube video [127] from the RailSem19 dataset is used. No frames that are used in the RailSem19 dataset are used in the proposed temporal dataset. Since the focus on the problem formulation of all temporal models is to improve the performance when driving over switches, the new dataset consists of short sequences dealing with this exact used case. All seconds in the video [127] are marked in which the train is located over a switch. Frames before and after these marked seconds are saved, resulting in 38 sequences with 76 frames each. These are annotated like the dataset used to train single-frame models.

Labeling 2888 images is very time-intensive and repetitive. Therefore, an auto-labeling strategy is implemented. The best-performing single-frame-based model in combination with the

adapted auto-crop method is utilized. Each image is processed 50 times so the auto-crop coordinates can adapt and only the most relevant region is considered. Figure 36 shows the process from this point. First, the mask's borders are taken, and then the horizon line is deleted leaving the lines of the rails. Only two pixels remain in each row as visualized in Figure 36c. These represent the final polylines. To minimize data only every fifth point is saved. This results in polylines consisting of roughly 62 points, which is the average number of points used to build polylines in the original dataset. Not all frames are correctly labeled because the temporal dataset focuses on scenes where the single-frame model is bound to fail. All incorrect annotated images are therefore manually edited in CVAT [126].

(a) (b) (c)

Figure 36: Process steps of auto-labeling: **(a)** prediction of best single-frame model and autocrop with 50 interations, **(b)** borders of prediction mask, **(c)** droped horizon line resulting in auto labeled polylines. All images are zoomed in for better visualization.

6.4 Data augmentation

In this section a thorough explanation of the data augmentation of [1] is given, for training and the cropping augmentation for inference. Additionally, all data augmentation strategies that are used in this work are described. This includes the methods adapted for training temporal models and an improved "Autocropper" technique for inference.

6.4.1 Data augmentation for Training single-frame-based models

[1] utilizes three data augmentation techniques in the training process. 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 [128] method from PyTorch is used. The third augmentation is the cropping mechanism, which reduces the image to the most relevant part. Randomness is also introduced to enhance generalization.

Figure 37 visualizes the cropping algorithm steps schematically. The GT is in the form of polylines and are represented by the two green lines. Then the yellow rectangle is computed, being the smallest possible rectangle around the GT. In the next step, the start pixels of the rails

GT are centered by expanding the yellow borders in one direction resulting in the orange box. All start pixels of rails are in the lowest row of an 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 predefined normal distribution. This introduced variability follows the policy, which ensures that the starting pixels of the rails at the bottom stay in the cropped image. Figure 37 illustrates four augmented variations of the top image in the bottom row.

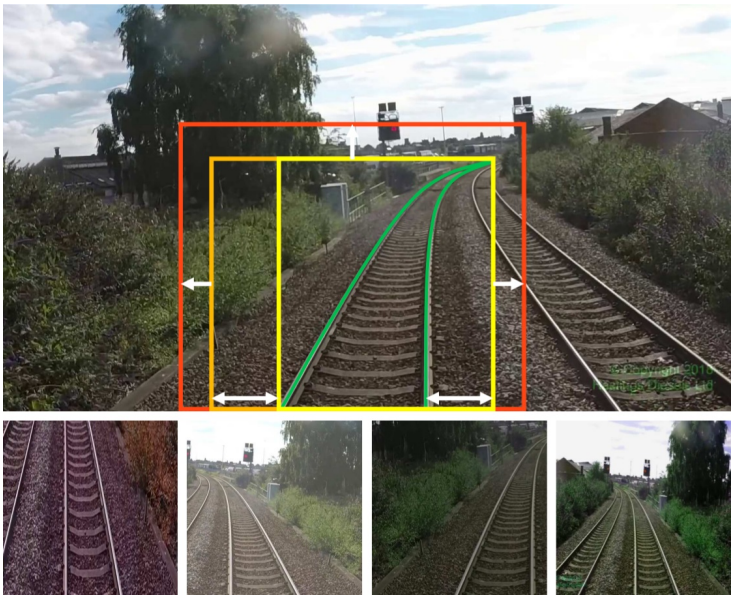


Figure 37: Data augmentation of TEP-Net [1], including the image horizontal flips, the color variations and the cropping mechanism.

6.4.2 Data augmentation for Training temporal models

For training temporal models the same three data augmentation strategies as the ones from training single-frame-based models are utilized for enhanced generalization. These again include horizontal flips, color variations, and a similar cropping mechanism. However, when training a model with sequences of images, the same augmentation must be used on all images of this sequence. For the first two augmentation techniques, random values are obtained and saved in variables. Then these random factors are applied to each image in the sequence. For the horizontal flip, a threshold at 0.5 and a random value between 0 and 1 define if a sequence is flipped or not. To achieve the same effect as the ColorJitter [128] function from the single-frame training, four uniform distributions are implemented outputting random values for brightness, contrast, saturation, and hue. These are then the parameters for the functions: `adjust_brightness` [129], `adjust_contrast` [130], `adjust_saturation` [131], and `adjust_hue` [132].

The cropping mechanism needs more consideration. First, the same method as the one

from single-frame training is applied to the first image of a sequence to obtain the red borders shown in Figure 37. To still guarantee randomness to minimize the risk of unwanted biases, the location of the left, right, and top borders are moved according to a normal distribution. To assure consistency along a sequence, the distances between the red borders and the random borders are calculated. From this point on the red borders are calculated in every image as before and these distances are used to incorporate random factors, instead of new random values for every image. The whole algorithm still follows a policy that the starting pixels of the rails stay in the image crop. Figure 38 visualizes an example sequence with raw images in the top row and augmented crops in the bottom row. The first, middle, and last images of a sequence with 10 images are shown. The train's progress is most clearly visible through the pole of the overhead line.

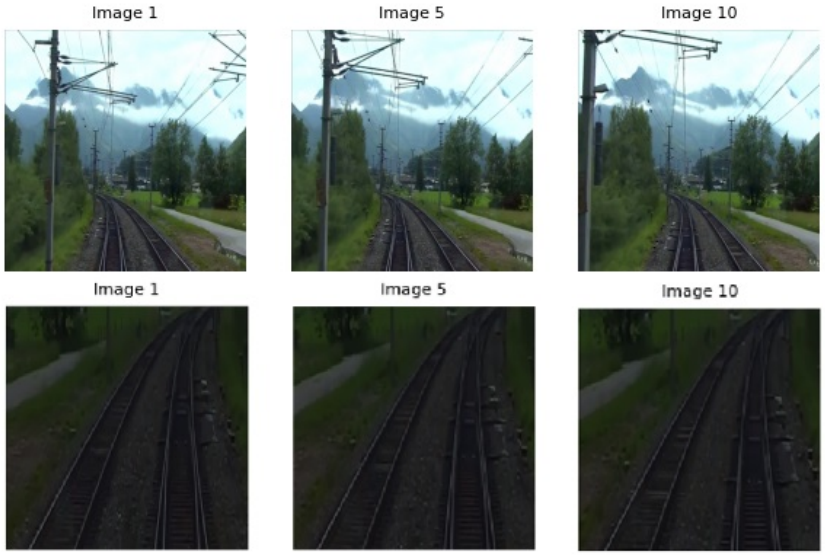


Figure 38: Data augmentation for temporal data processing, including the 1st, 5th and 10th image of a sequence. First row resized raw images. Second row augmented images with the same three augmentation techniques for the whole sequence.

6.4.3 Cropping in inference with TEP's Autocrop

In [1] only cropped images are used for training models to only consider the most relevant Region of Interest (ROI) in images. 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 39 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 coordinates 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. Furthermore, the global averages of crop coordinates are taken in consideration. This allows the crop coordinates to converge to these global averages and prevent collapse scenarios, in which the crop becomes to small.

Figure 39: Example of perspective shifts with three different scenarios: left curve, right curve and straight rail [1].

6.4.4 Improved Autocrop for inference

[1] incorporates a Running Average (RA) Equation 3 with rule-based policies to prevent collapse to the inside and consider image borders when adding a margin. After that, the newly calculated and checked average is weighted and added to form the new crop coordinates, like in Equation 4. In these formulas AVG is the average, x is the crop coordinates and $pred$ is the values obtained from the prediction. In Equation 4 the same annotations are used with c being the coefficient, which is 0.1. All notations but c include all three crop coordinates: left, top right.

$$AVG_t = \frac{AVG_{t-1} \cdot (t-1) + pred_t}{t} \tag{3}$$

$$x_t = x_{t-1} * (1-c) + AVG_t * c \tag{4}$$

However, observations in the inference of videos demonstrate that the introduced autocrop mechanism is not optimal. The behavior shows that the RA reacts slowly to the inside and the

outside. This is especially problematic when the crop should quickly expand in one direction. Additionally, the borders of the crop converge to an overall average, presenting the same issues as when simply pre-defining a crop. Those behaviors can push the prediction outside the rails. In more detail when the rails move to the left or right in the image due to a perspective shift when the train takes a curve and the Autocroper does not react quickly enough, the rails are subsequently positioned outside the crop area. An additional issue presents the lack of an recovery option when the image crop is completely incorrect.

$$x_t = x_{t-1} * (1 - c) + pred_t * c \tag{5}$$

Therefore a new algorithm is implemented for this work to present a more robust solution. The desired behavior should be one that is more responsive to the outside to react to rapid changes in perspective. Simultaneously, the crop should converge gradually to the inside to prevent any fast movements of crop coordinates, which possibly lead to collapse events.

To realize such an algorithm an Exponential Moving Average (EMA) is utilized shown in Equation 5, in which only the old crop coordinates and the values from the prediction are weighted and added. c is the coefficient which is set to 0.01 presenting slow movements to the inside. Additionally, with the same collapse prevention technique as before the crop cannot be smaller than the smallest rectangle of the prediction (yellow box in Figure 37). However, by directly calculating the crop coordinates without any averages in between like Equation 3, this leads to a different behavior. The collapse prevention rule now overwrites the coordinates when the prediction expands, resulting in a rapid widening of the crop.

Furthermore, the crop is bound to stay within image borders. Further adaptations to enhance robustness are done by increasing the margins, which are added to the prediction rectangle (yellow box in Figure 37) to give the model more leeway.

Additionally, a reset rule is implemented in which the crop coordinates are overwritten with predefined values. The behavior is observed that when models become uncertain, the horizon line converges to the bottom of the image. This behavior is used and the crop resets when the predicted horizon line is below 40% of the crop height. Experiments show that the following reset values are the most advantageous: left = 1/3 of image width, top = 1/2 of image height, right = 2/3 of image width.

6.5 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. Since, this loss function demonstrates to be functional, it is adopted unchanged in this work for training single-frame-based models and temporal models. This loss function Equation 6 [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.

$$L = L_{traj} + \lambda \times L_{ylim} \tag{6}$$

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 7. This is done with a $\beta_1 = 0.005$. As shown in figure Figure 40, 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 counteract 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 7.

$$L_{traj} = \frac{\sum_{i=1}^{H} m_i \times w_i \times L_{rails}(\hat{\mathbf{X}}_i, \mathbf{X}_i)}{\sum_{i=1}^{H} m_i} \tag{7}$$

$$L_{rails} = \sum_{r \in \{left, right\}} SmoothL1(\hat{\mathbf{X}}_{i,r} - \mathbf{X}_{i,r}, \beta_1) \tag{8}$$

$$SmoothL1(x, \beta) = \begin{cases} 0.5x^2/\beta, & \text{if } |x| < \beta \\ |x| - 0.5 * \beta, & \text{otherwise} \end{cases} \tag{9}$$

$$w_i = \min\left(\frac{1}{\mathbf{X}_{i,right} - \mathbf{X}_{i,left}}, W_{max}\right) \tag{10}$$

$$m_i = \begin{cases} 1 & \text{if } i \leq y_{lim} \times H \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

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 of 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.

$$L_{ylim} = SmoothL1(\hat{y_{lim}} - y_{lim}, \beta_2) \tag{12}$$

Figure 40 shows a plot of the Equation 9 being the $SmoothL1$ Loss function which is widely used in the literature.

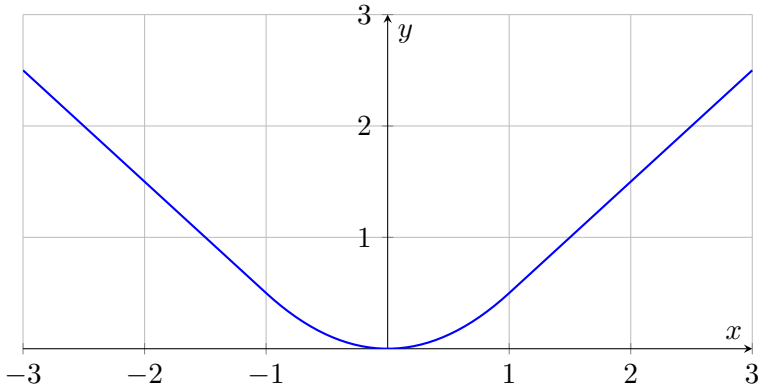


Figure 40: $SmoothL1$ Loss Function

6.6 Setup used for Training CNNs

For training CNNs 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 [133]. 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 [134]. The server employs two Intel(R) Xeon(R) Gold 5118 CPUs @ 2.30GHz [135]. These Central Processing Unit (CPU)s have 24 threads and 12 cores each.

All trainings are logged with the "Weights & Biases" [136], a developer platform for training and fine-tuning machine learning models. [136] 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" [136] 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.

6.7 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 [137]. 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 [138].

6.7.1 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 [139]. 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 [140]. 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 9.

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 9: Jetson AGX Xavier technical specifications [139]

6.7.2 Optimizing models with TensorRT

To fully leverage the used hardware platform Jetson AGX Xavier, NVIDIA introduced TensorRT [141]. 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] [141]. 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 [141]. 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 [142]. 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 [142]. 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 [142]. 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 [143]. 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

blidtext

7.1 Object detection Experiments

The first approach pursued in this work is to combine an object detection model and a semantic segmentation model. The semantic segmentation model should predict the track on a pixel level, and the object detection model predicts the direction of the train in scenarios where switches divide the track.

Experiments started with training object detection models from the YOLO series. The models YOLOv7 [44] and YOLOv9 [45] are selected to investigate their ability to predict switches and their states. Experiments have been conducted with RailSem19 and subsets of this dataset, which are further described in subsection 6.3.1. At the time YOLOv9 has just been released. Consequently, not all models were fully supported during the experiments. Therefore, the versions GELAN-c and GELAN-e are utilized. For the YOLOv7 the smallest version with the highest reported speed is used. Many experiments are conducted with these three models, however only the best-performing are described.

All YOLO models have built-in data augmentation. This includes a horizontal flip of images. Since this horizontal flip of images only flips images but does not change the label $switch\_left$ to $switch\_right$ and vice versa, this strategy cannot be used. Consequently, the probability of such flips is defined to be zero. The remaining default data augmentation is utilized in all experiments. Additionally, all other default hyperparameters are not changed, and parameters that are specified in the training command are used as they are recommended in the GitHub repositories [117] and [116]. All changed and specified parameters of the best-performing experiments are shown in Table 10. Furthermore, all models come with pre-trained weights trained on the COCO dataset. In all experiments, these weights are used as initialization.

Parameters	YOLOv9 models	YOLOv7
epochs	500	500
fliplr	0.0	0.0
batch_size	16	16
img_size	640	640
workers	8	8
min-items	0	
close-mosaic	15	

Table 10: Used parameters for training YOLO object detection models

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] Z. Wang, X. Wu, G. Yu, and M. Li, "Efficient rail area detection using convolutional neural network," *IEEE Access*, vol. 6, pp. 77 656–77 664, 2018. DOI: 10.1109/ACCESS.2018.2883704.

[48] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. DOI: 10.1109/TPAMI.2016.2644615.

[49] D. Yu, H. Wang, P. Chen, and Z. Wei, “Mixed pooling for convolutional neural networks,” in *Rough Sets and Knowledge Technology*, D. Miao, W. Pedrycz, D. Ślezak, G. Peters, Q. Hu, and R. Wang, Eds., Cham: Springer International Publishing, 2014, pp. 364–375, ISBN: 978-3-319-11740-9.

[50] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018. DOI: 10.1109/TPAMI.2017.2699184.

[51] 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.

[52] J. Long, E. Shelhamer, and T. Darrell, “ Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298965.

[53] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, “Full-resolution residual networks for semantic segmentation in street scenes,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3309–3318. DOI: 10.1109/CVPR.2017.353.

[54] H. Li, Q. Zhang, D. Zhao, and Y. Chen, “Railnet: An information aggregation network for rail track segmentation,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9206967.

[55] 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.

[56] O. Katar and E. Duman, “Automated semantic segmentation for autonomous railway vehicles,” *Tehnički Glasnik*, vol. 16, pp. 484–490, Sep. 2022. DOI: 10.31803/tg-20220329114254.

[57] 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.

[58] 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*, M. El Yacoubi, E. Granger, P. C. Yuen, U. Pal, and N. Vincent, Eds., Cham: Springer International Publishing, 2022, pp. 185–197, ISBN: 978-3-031-09037-0.

[59] 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.

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

[61] T. Wu, S. Tang, R. Zhang, J. Cao, and Y. Zhang, “Cgnet: A light-weight context guided network for semantic segmentation,” *IEEE Transactions on Image Processing*, vol. 30, pp. 1169–1179, 2021. DOI: 10.1109/TIP.2020.3042065.

[62] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 833–851, ISBN: 978-3-030-01234-2.

[63] R. Barth, J. IJsselmuiden, J. Hemming, and E. Van Henten, “Synthetic bootstrapping of convolutional neural networks for semantic plant part segmentation,” *Computers and Electronics in Agriculture*, vol. 161, pp. 291–304, 2019, BigData and DSS in Agriculture, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2017.11.040. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917307664.

[64] Z. Tao, S. Ren, Y. Shi, X. Wang, and W. Wang, “Accurate and lightweight railnet for real-time rail line detection,” *Electronics*, vol. 10, no. 16, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10162038. [Online]. Available: https://www.mdpi.com/2079-9292/10/16/2038.

[65] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, “Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation,” *International Journal of Computer Vision*, vol. 129, no. 11, pp. 3051–3068, Sep. 2021, ISSN: 1573-1405. DOI: 10.1007/s11263-021-01515-2. [Online]. Available: http://dx.doi.org/10.1007/s11263-021-01515-2.

[66] Z. Chen, W. Niu, C. Wu, L. Zhang, and Y. Wang, “Near real-time situation awareness and anomaly detection for complex railway environment,” in *2021 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, 2021, pp. 1–8. DOI: 10.1109/CogSIMA51574.2021.9475947.

[67] S. Belyaev, I. Popov, V. Shubnikov, P. Popov, E. Boltenkova, and D. Savchuk, "Railroad semantic segmentation on high-resolution images," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294722.

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 346–361, ISBN: 978-3-319-10578-9.

[69] 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.

[70] M. Ghorbanalivakili, J. Kang, G. Sohn, D. Beach, and V. Marin, "Tpe-net: Track point extraction and association network for rail path proposal generation," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, 2023, pp. 1–7. DOI: 10.1109/CASE56687.2023.10260541.

[71] 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.

[72] K. Zhou, *Tusimple-benchmark*, GitHub repository, 2017. [Online]. Available: https://github.com/TuSimple/tusimple-benchmark/tree/master.

[73] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[74] K. Zhou, *Tusimple-benchmark*, GitHub repository, 2017. [Online]. Available: https://github.com/TuSimple/tusimple-benchmark/blob/master/doc/lane_detection/assets/examples/lane_example.jpg.

[75] F. Pizzati, M. Allodi, A. Barrera, and F. García, "Lane detection and classification using cascaded cnns," in *Computer Aided Systems Theory–EUROCAST 2019: 17th International Conference, Las Palmas de Gran Canaria, Spain, February 17–22, 2019, Revised Selected Papers, Part II 17*, Springer, 2020, pp. 95–103.

[76] S. Yoo, H. Seok Lee, H. Myeong, S. Yun, H. Park, J. Cho, and D. Hoon Kim, "End-to-end lane marker detection via row-wise classification," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 4335–4343. DOI: 10.1109/CVPRW50498.2020.00511.

[77] X. Li, J. Li, X. Hu, and J. Yang, "Line-cnn: End-to-end traffic line detection with line proposal unit," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 248–258, 2020. DOI: 10.1109/TITS.2019.2890870.

[78] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, " Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2021, pp. 294–302. DOI: 10.1109/CVPR46437.2021.00036. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00036.

[79] H. Xu, S. Wang, X. Cai, W. Zhang, X. Liang, and Z. Li, "Curvelane-nas: Unifying lane-sensitive architecture search and adaptive point blending," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 689–704, ISBN: 978-3-030-58555-6.

[80] J. Wang, Y. Ma, S. Huang, T. Hui, F. Wang, C. Qian, and T. Zhang, "A keypoint-based global association network for lane detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1392–1401.

[81] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*, Springer, 2020, pp. 276–291.

[82] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, " PolyLaneNet: Lane Estimation via Deep Polynomial Regression," in *2020 25th International Conference on Pattern Recognition (ICPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jan. 2021, pp. 6150–6156. DOI: 10.1109/ICPR48806.2021.9412265. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICPR48806.2021.9412265.

[83] Z. Feng, S. Guo, X. Tan, K. Xu, M. Wang, and L. Ma, *Rethinking efficient lane detection via curve modeling*, 2023. arXiv: 2203.02431 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2203.02431.

[84] S. Yang, G. Yu, Z. Wang, B. Zhou, P. Chen, and Q. Zhang, "A topology guided method for rail-track detection," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, pp. 1426–1438, 2022. DOI: 10.1109/TVT.2021.3133327.

[85] 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.

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

[87] 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.

[88] 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).

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

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

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

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

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

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

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

[104] 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.

[105] 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.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

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

[115] 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.

[116] WongKinYiu, *Yolov9*, GitHub repository, 2024. [Online]. Available: https://github.com/WongKinYiu/yolov9.

[117] ——, *Yolov7*, GitHub repository, 2022. [Online]. Available: https://github.com/WongKinYiu/yolov7.

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

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

[120] T. Laurent, *Train-ego-path-detection*, GitHub repository, 2024. [Online]. Available: https://github.com/irtrailenium/train-ego-path-detection.

[121] PyTorch, *Adaptiveavgpool2d*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html (visited on 11/29/2024).

[122] ——, *Adaptivemaxpool2d*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveMaxPool2d.html (visited on 11/29/2024).

[123] ——, *Relu*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html (visited on 11/29/2024).

[124] ——, *Torch.tensor.view*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.Tensor.view.html (visited on 11/30/2024).

[125] ——, *Onecyclelr*, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html (visited on 12/01/2024).

[126] CVAT, *Open data annotation platform*, 2024. [Online]. Available: https://www.cvat.ai/ (visited on 12/01/2024).

[127] W. YouTube, *Austria salzburg villach*, 2024. [Online]. Available: https://www.youtube.com/watch?v=1nj7kmw2G2M&t=6973s (visited on 12/02/2024).

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

[129] ——, *Adjust_brightness*, 2024. [Online]. Available: https://pytorch.org/vision/0.19/generated/torchvision.transforms.functional.adjust_brightness.html (visited on 11/27/2024).

[130] ——, *Adjust_contrast*, 2024. [Online]. Available: https://pytorch.org/vision/0.19/generated/torchvision.transforms.functional.adjust_contrast.html (visited on 11/27/2024).

[131] ——, *Adjust_saturation*, 2024. [Online]. Available: https://pytorch.org/vision/0.19/generated/torchvision.transforms.functional.adjust_saturation.html (visited on 11/27/2024).

[132] ——, *Adjust_hue*, 2024. [Online]. Available: https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_hue.html (visited on 11/27/2024).

[133] 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).

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

[135] 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).

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

[137] 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).

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

[139] ——, *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).

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

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

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

[143] 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 Track segmentation in front of the train. Red are all adjacent tracks and green are possible tracks [67]. 12
Figure 7 Example images of TuSimple with annotations [74]. Red lines are horizontal anchor lines. Green circles mark image coordinates, where road lanes and anchor lines intersect. 13
Figure 8 Rail detection with Gridsystem [85]. Hyperparameters: C is the number of rails (number of grids). h is the number of cells vertically. $w + 1$ is the number of cells horizontally with an additional column for a background class, when no rail is detected in a row. 15
Figure 9 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. 17
Figure 10 Comparison between classification (CLS), regression (REG) and segmentation (SEG) models with challenging scenes. The worst backbone (ResNet18) is used for this figure to clearly show difference in behaviours [1]. 18
Figure 11 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 [86]. 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. 19
Figure 12 ResNet residual block [51] 21
Figure 13 DenseNet dense block [71] 22

Figure 14 MobileNet V2 and V3 Blocks [93]: **(a)** MobileNetV2 bottleneck block, **(b)** MobileNetV3 bottleneck block with added Squeeze-and-Excite block 23
Figure 15 Systematic visualisation of scaling parameters of a network [94] 24
Figure 16 RailSem19 dataset examples. First row raw images. Second row dense GT [26]. 30
Figure 17 Example images and GT of RailVID dataset [60] 30
Figure 18 RailSet-Seg example with annotations [59] [58]: **(a)** raw-image, **(b)** rail class, **(c)** rail and rail-track class 31
Figure 19 RSDS example with annotation [27]: **(a)** raw-image, **(b)** ground truth 32
Figure 20 Rail-DB [85] images with annotations in different conditions 33
Figure 21 OSDaR23 data from all different sensors [115] 34
Figure 22 OSDaR23 annotated scene [115] 34
Figure 23 Examples images from RailSem19, which are not included in the TEP-Net dataset due to unclear circumstances about the trains direction. [1] 35
Figure 24 TEP-Net dataset example images with annotation [1] 35
Figure 25 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. 37
Figure 26 Backbone logic (Backbone graphic taken from [1]). 39
Figure 27 Integration of a true Pooling Layer (Model graphic taken from [1] and changed). 40
Figure 28 Different architectures for prediction heads. 41
Figure 29 $ReLU$ Activation Function [123] 42
Figure 30 Schematic drawing of a temporal model with an integrated RNN module (Graphic taken from [1] and changed). 42
Figure 31 8 different temporal models for experiments. **(c)** and **(e)** have two versions in which the number of FC layers vary. One FC layer is used in for V1 and three Layers are used for the V2 model. Models **(a)**, **(b)**, **(c)** and **(d)** include LSTMs modules. Models **(e)** and **(f)** try to learn temporal dependencies only with FC layers. Shape of heads resemble the architecture e.g. trapeze shaped contour is the trapeze head shown in Figure 28d 43
Figure 32 Comparison between data handling processes for single-frame training and the proposed temporal training. 45
Figure 33 Sampler is responsible for the sliding window approach in temporal training: bottom lines represents the sequence with a length of 76 frames; the box is the window of frames used as the input for the model with x being the length of this window (with $x \leq 76$); the blue part of the sliding window represents the output frame, because this temporal method deals with a sequence-to-one problematic. 45
Figure 34 Example of a switch scene that is incorrectly predicted, but the IoU metric still returns high values presenting potentially misleading performance scores. Track continuous to the left. 48

Figure 35 Scoring system of the switch evaluation dataset. 49
Figure 36 Process steps of auto-labeling: **(a)** prediction of best single-frame model and autocrop with 50 interations, **(b)** borders of prediction mask, **(c)** droped horizon line resulting in auto labeled polylines. All images are zoomed in for better visualization. 50
Figure 37 Data augmentation of TEP-Net [1], including the image horizontal flips, the color variations and the cropping mechanism. 51
Figure 38 Data augmentation for temporal data processing, including the 1st, 5th and 10th image of a sequence. First row resized raw images. Second row augmented images with the same three augmentation techniques for the whole sequence. 52
Figure 39 Example of perspective shifts with three different scenarios: left curve, right curve and straight rail [1]. 53
Figure 40 $SmoothL1$ Loss Function 56

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 27
Table 4 Datasets for Semantic Segmentation 28
Table 5 Versions of backbones utilized in this work 38
Table 6 Versions of backbones utilized in this work 39
Table 7 Detailed network structure of Trapeze Head 41
Table 8 Used dataset subsets of RailSem19 for training YOLO object detection models . 47
Table 9 Jetson AGX Xavier technical specifications [139] 58
Table 10 Used parameters for training YOLO object detection models 62

Quellcodeverzeichnis

1 Hello-World 3
2 Exporting a PyTorch model to ONNX format 59

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

ASPP Atrous Spatial Pyramid Pooling

SPP Spatial Pyramid Pooling

FPS Frames Per Second

TPE-Net Track Point Extraction and Association Network

CGNet Context Grided Network

BiSeNet Bilateral Semantic segmentation Net

NAS Network Architecture Search

RA Running Average

EMA Exponential Moving Average

FC Fully Connected

ReLU Rectified Linear Unit

A Anhang A

B Anhang B