Machine Learning-Based Radar Point Cloud Segmentation

Segmentierung von Radar Punktwolken mit maschinellen Lernverfahren

Scientific work for obtaining the academic degree

Master of Science (M.Sc.)

at the Department of Mechanical Engineering of Technical University of Munich

Supervised by Prof. Dr.-Ing. Markus Lienkamp
Felix Nobis, M.Sc.
Chair of Automotive Technology

Submitted by Felix Fent, B.Sc.

Submitted on 28.10.2020

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Aufgabenstellung

Machine Learning-Based Radar Point Cloud Segmentation

Hochautomatisierte Fahrzeuge stellen sicherheitskritische Systeme dar, deren Funktion in wechselnden Umweltbedingungen sichergestellt sein muss. Die funktionale Sicherheit dieser Fahrzeuge beruht unter anderem auf zuverlässigen und redundanten Sensorsystemen zur Wahrnehmung der Umgebung. Um diesen Anforderungen gerecht zu werden sind radarbasierte Systeme, aufgrund deren Zuverlässigkeit in unterschiedlichen Umweltbedingungen, Gegenstand aktueller Forschung.

In dieser Arbeit sollen Segmentierungsverfahren für radarbasierte Punktwolken für das hochautomatisierte Fahren entwickelt werden. Aufgrund des Radarprinzips stehen bei der Entwicklung geeigneter Methoden temporale Zusammenhänge im Fokus. Das Ziel der vorliegenden Arbeit ist bestehende Radarsegmentierungsverfahren auf einem öffentlich zugänglichen Datensatz zu vergleichen. Darüber hinaus soll ein rekurrentes Modell zur Segmentierung von Radarpunktwolken unter Einbezug der Zeitdimension entwickelt werden.

In einem ersten Schritt soll anhand einer Literaturrecherche und bisherigen Arbeiten am Lehrstuhl der Stand der Technik zur Segmentierung von Radardaten erarbeitet werden. Darauf aufbauend soll ein Modell zur Segmentierung von Radarpunkten entwickelt werden. Der Fokus dieses Modells liegt auf der Verwendung temporaler Zusammenhänge. Abschließend soll das Modell auf einem öffentlichen Datensatz evaluiert und die Güte der Segmentierung bewerten werden.

Folgende Punkte sind durch Herrn Felix Fent zu bearbeiten:

- Einarbeitung in den Stand der Technik der maschinellen Lernverfahren zur Segmentierung von Radarpunktwolken.
- Identifizierung eines geeigneten Datensatzes zur Modellerstellung.
- Analyse und Vergleich bestehender Modellansätze zur Segmentierung von Radardaten.
- Entwicklung eines rekurrenten neuronalen Netzes zur Segmentierung von Radarpunktwolken.
- Evaluation der Segmentierungsgüte des Modells auf einem öffentlich zugänglichen Datensatz.

Die Ausarbeitung soll die einzelnen Arbeitsschritte in übersichtlicher Form dokumentieren. Der Kandidat verpflichtet sich, die Masterarbeit selbständig durchzuführen und die von ihm verwendeten wissenschaftlichen Hilfsmittel anzugeben.

Die eingereichte Arbeit verbleibt als Prüfungsunterlage im Eigentum des Lehrstuhls.

Ausgabe: 04.05.2020	Abgabe: 28.10.2020
______________________	______________________
Prof. Dr.-Ing. M. Lienkamp	Betreuer: Felix Nobis, M. Sc.

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Geheimhaltungsverpflichtung

Herr: Fent, Felix

Gegenstand der Geheimhaltungsverpflichtung sind alle mündlichen, schriftlichen und digitalen Informationen und Materialien, die der Unterzeichner vom Lehrstuhl oder von Dritten im Rahmen seiner Tätigkeit am Lehrstuhl erhält. Dazu zählen vor allem Daten, Simulationswerkzeuge und Programmcode sowie Informationen zu Projekten, Prototypen und Produkten.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere:

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des Betreuers an Dritte weiterzuleiten,
- ohne Zustimmung eines Mitarbeiters keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an den Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Für den Dateiaustausch dürfen keine Dienste verwendet werden, bei denen die Daten über einen Server im Ausland geleitet oder gespeichert werden (Es dürfen nur Dienste des LRZ genutzt werden (Lehrstuhllaufwerke, Sync&Share, GigaMove).
- Vertrauliche Informationen dürfen nur in verschlüsselter Form per E-Mail versendet werden.
- Nachrichten des geschäftlichen E-Mail Kontos, die vertrauliche Informationen enthalten, dürfen nicht an einen externen E-Mail Anbieter weitergeleitet werden.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 04.05.2020

Unterschrift: ___________________________________

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 28.10.2020

____

Felix Fent, B. Sc.

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Declaration of Consent, Open Source

Hereby I, Fent, Felix, born on 04.05.1996, make the software I developed during my master thesis available to the Institute of Automotive Technology under the terms of the license below.

Garching, 04.05.2020

____

Felix Fent, B. Sc.

Table of Contents

List of Abbreviations

ANN	Artificial neural network
CAN	Controller area network
CNN	Convolutional neural network
E/E-System	Electric/Electronic-System
FP	Feature propagation
GNN	Graph neural network
KPConv	Kernel point convolution
KP-FCNN	Kernel point fully convolutional neural network
KPLSTM	Kernel point convolutional long short-term memory
Lidar	Light detection and ranging
LSTM	Long short-term memory
MLP	Multilayer perceptron
nuScenes	nuTonomy scenes
Radar	Radio detection and ranging
RCS	Radar cross section
ReLU	Rectified linear unit
ResNet	Residual network
RNN	Recurrent neural network
SA	Set abstraction
SVM	Support vector machine

Formula Symbols

Formula symbols	Unit	Description
a		Alpha compensation factor
b		Batch size (number of batched scenes)
$\mathcal{B}_{\mathrm{r}}$		Neighborhood domain
$\mathbf{c}$		Class weights vector
$\mathbf{C}$		Cell state
$\tilde{\mathbf{C}}$		Internal cell state
$\hat{\mathbf{C}}$		Intermediate cell state
$\mathcal{C}$		Finite set of point coordinates
d		Number of dimensions (radar channels)
dr		Decay rate of the learning rate
ds		Decay steps of the learning rate decay
$\mathbf{f}$		Feature vector of a point
f		Arbitrary function
fn		Number of false negatives
fp		Number of false positives
$\mathbf{F}$		Forget gate
$\mathrm{F1}$		F1 score
$\mathcal{F}$		Finite set of point features
g		Arbitrary kernel function
h		Correlation function
$\mathbf{H}$		Hidden state
$\mathbf{I}$		Input gate
k		Number of classes (labels)
log		Logarithm

lr	Learning rate
lr_0	Initial learning rate
$\mathcal{L}$	Loss function
max	Maximum value of a radar channel
max	Maximum operation
min	Minimum value of a radar channel
min	Minimum operation
n	Number of points (elements of the set)
n_k	Number of kernel points
$\mathcal{N}_\mathbf{x}$	Set of points in the local neighborhood of **x**
$\mathbf{O}$	Output gate
$\mathbf{p}$	Point as element of a point cloud
p_θ	Modified PointNet++ layer
$\mathcal{P}$	Point cloud as a finite set of points
$\mathcal{P}_j'$	Set of points that belong to class j
r	Radius of the local neighborhood
r_0	Initial radius of the local neighborhood
$\tilde{\mathrm{r}}$	Relation between all points (scene representation)
$\mathbb{R}$	Set of all real numbers
s	Maximum number of keyframes per scene
$tanh$	Hyperbolic tangent
tp	Number of true positives
$\mathbf{w}$	Kernel weight
$\mathbf{x}$	Coordinate vector of a point
$\bar{\mathbf{x}}$	Relative position of a point to its center point
$\tilde{\mathbf{x}}$	Kernel point
$\mathbf{X}$	Model input data tensor
$\acute{\mathbf{X}}$	Structured input data of a fixed size
$\boldsymbol{y}$	Target value vector of a point (label)
$\mathbf{Y}$	Model target value tensor (labels)
$\hat{\mathbf{Y}}$	Output tensor of the model (prediction)

$\mathcal{Y}$	Set of target values (labels)
α	Balancing factor
γ	Focus factor
ε	Small value different from zero
Θ, ψ	Parameters of the model
σ	Area of influence of the kernel points
σ	Sigmoid function
$\circ$	Composite of two functions
$*$	Convolution operator
$\odot$	Hadamard product
$\lvert\cdot\rvert$	Cardinality of a set
$\lVert\cdot\rVert$	Euclidean norm

1 Introduction

1.1 Motivation

Autonomous driving, as one of the greatest challenges in the automotive industry today, relies on the perception and understanding of the environment around the autonomous vehicle [1]. This understanding of the environment is essential to operate safely even within complex traffic scenarios and challenging weather conditions. Thus, the functional safety of autonomous vehicles is subject to reliable sensor information and redundant information processing [2].

The reliability of the sensor information must be ensured under any environmental conditions, which is difficult to accomplish with a single sensor type. The informational content of a camera sensor, for example, is highly affected by severe weather conditions like rain and fog as well as rapid changes in brightness or direct illumination – as illustrated in Figure 1.1 [3]. To overcome these limitations, radio detection and ranging (radar) sensors can be applied, which are mostly unaffected by environmental conditions [4]. In addition to that, radar sensors provide highly abstract information about the surrounding objects such as their distance or relative velocity at a significantly smaller bandwidth than pixel-based camera sensors [5].

Besides reliable sensor information, a redundant process chain is required to minimize the risk of hazards caused by malfunctioning behavior of electric/electronic systems (E/E-Systems) [6]. This demand on a fully redundant process chain makes it indispensable to develop non-camera-based models to sense the environment around the autonomous vehicle. Driven by the demand for radar-based model architectures and motivated by the most recent achievements in point cloud processing, it is the aim of this thesis to develop a machine learning-based model for radar point cloud segmentation.

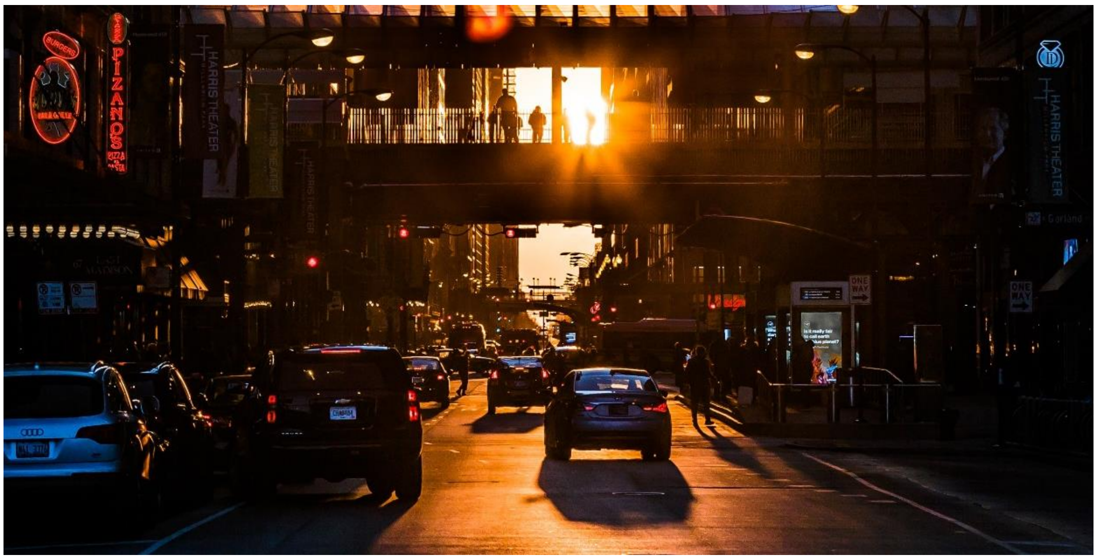


Figure 1.1: Illustration of a typical traffic scenario with difficult lighting conditions by [7].

1.2 Research Purpose

The development of a radar point cloud segmentation model is built upon the method of CHOLLET [8, pp. 111-116], which consists of four major modeling steps. These modeling steps are illustrated in Figure 1.2 and associated with the stated research questions, as discussed in the subsequent paragraphs.

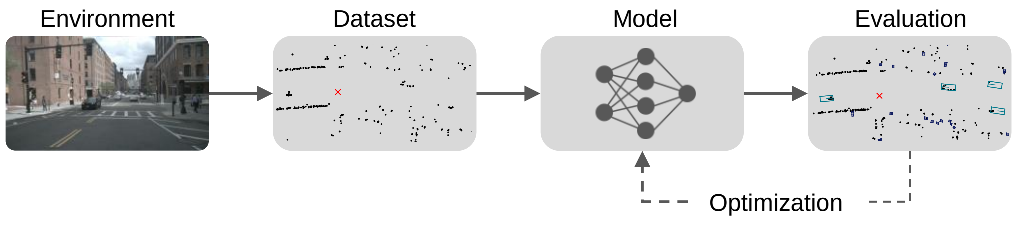


Figure 1.2: Illustration of the four major modeling steps according to CHOLLET [8].

The selection (or acquisition) of the dataset to train the model defines the first modeling step and builds the basis for all subsequent development stages. Therefore, the dataset must include both the model input data as well as the aspired output data to evaluate the model's performance. Within the scope of this thesis, this modeling step is associated with the following research question.

How do state of the art models perform on publicly available datasets?

In the second modeling step a suitable evaluation metric is defined to measure the model's performance and evaluate the success on the given problem. This step also includes the definition of an evaluation procedure that ensure the objectivity of the evaluation process and is subject to the following research question.

How to evaluate different segmentation models and settings?

According to a data driven development strategy the model is developed based on a given dataset in the third modeling step. The development of the model architecture – which represents the core of this thesis – defines both the fundamental properties of the model as well as the model behavior and will be discussed with focus on the following research question.

Does the utilization of the temporal domain enhance the segmentation quality?

The final modeling step – the model optimization – can be described as the process of adjusting parameters to best approximate the aspired transmission behavior between the model input data and the desired output. However, within the scope of this thesis the model optimization process serves the purpose of exploring the parameter space to discuss the following research question.

How do different parameters affect the segmentation quality of the model?

Under consideration of the above defined method, the stated research questions are subject to the following constraints. First, the acquisition of a dataset is not part of the thesis but limited to the selection of a suitable publicly available dataset. Second, the segmentation model will be implemented as an artificial neural network (ANN), not considering other machine learning approaches. Given these constraints, it is the purpose of the thesis to develop a radar point cloud segmentation model and to discuss the stated research questions.

1.3 Structure of the Thesis

The macroscopic structure of the thesis represents a data driven development process and is illustrated in Figure 1.3. This process is chosen in accordance with the defined method of CHOLLET [8, PP. 111-116] and in consideration of the research questions stated in section 1.2. The same sequence of steps is applied to every individual chapter to develop a model built upon the given data and not vice versa.

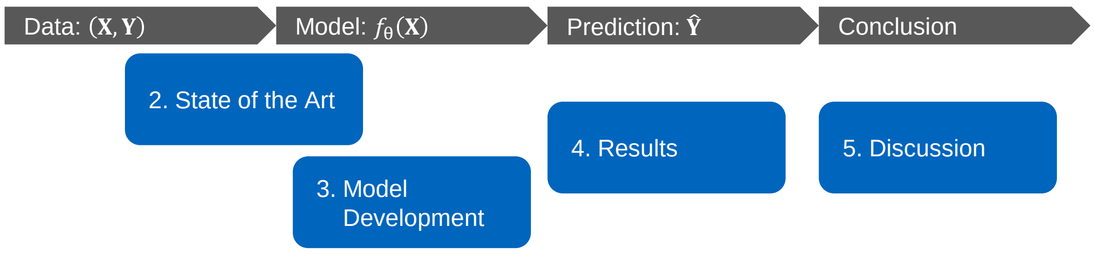

Data: $(\mathbf{X}, \mathbf{Y})$
Model: $f_\theta(\mathbf{X})$
Prediction: $\hat{\mathbf{Y}}$
Conclusion
2. State of the Art
3. Model Development
4. Results
5. Discussion

Figure 1.3: Graphical representation of the structure of the thesis following a data driven development strategy.

The research purpose of this thesis is motivated by the demand for machine learning-based models used for the segmentation of automotive radar data and defined in section 1.2. Based on these research questions, the current state of the art in radar point cloud segmentation is described in Chapter 2. First, an overview of publicly available datasets for autonomous driving is given in section 2.1 from which the nuScenes dataset is selected as basis of the thesis and further described in section 2.2. Regarding this dataset, a definition of the data format is provided in section 2.3, before different model architectures are discussed in section 2.4. These model architectures are divided into feedforward neural networks in subsection 2.4.1 and recurrent neural networks in subsection 2.4.2.

In consideration of the current state of the art, a novel model architecture for radar point cloud segmentation is proposed in Chapter 3. The chapter is structured in accordance with the defined method of CHOLLET [8] and represents a data driven development process. The data pre-processing is described in section 3.1, which builds the basis for the data input pipeline of section 3.2. Building upon this data pipeline, a recurrent neural network for point cloud segmentation is developed in section 3.3. To utilize the proposed network architecture a loss function is defined in section 3.4, which is minimized throughout the model training. This optimization process is executed by an optimizer defined in section 3.5 and the success of the optimization process is evaluated according to the evaluation metric of section 3.6. The overall model training is described in section 3.7 to define the experimental design of the thesis.

The results of the conducted experiments are presented in Chapter 4 and structured in accordance with the stated research questions. Following the same structure, the model results as well as the model architecture itself are discussed in Chapter 5. The experimental design is discussed in section 5.1 with respect to the quality criteria of empirical research, and the chosen nuScenes dataset is discussed in section 5.2. On this basis, the results of the feedforward and recurrent neural networks are discussed in section 5.3 and section 5.4, respectively. Ultimately, the findings of the thesis are summarized in section 6.1 and an outlook on future research is given in section 6.2.

2 State of the Art

Following a data driven development strategy, a comparison of publicly available datasets for autonomous driving is given first in section 2.1. As a result of this comparison, the nuScenes dataset is chosen as data basis for this work and further described in section 2.2. Following that, the data format of point cloud data is defined in section 2.3, introducing the notation used throughout the thesis. Finally, state of the art network architectures for radar point cloud data are presented in section 2.4, which lays the foundation for the model development.

2.1 Datasets for Automomous Driving

Machine learning-based radar point cloud segmentation describes the process of building a mathematical model based on sample data in order to assign a class label to every point of the radar point cloud. To achieve this goal by a supervised machine learning approach the dataset must contain both the radar point cloud as well as the associated target values. In order to find a dataset that satisfies these requirements, a comparison of different publicly available datasets for autonomous driving is given in Table 2.1.

Table 2.1: A comparison of publicly available datasets for autonomous driving with focus on their contained sensor information. An empty circle represents the absence of this data and a filled circle represents the containment of the data.

Dataset	Number of samples	Number of scenes	Camera	Lidar	Radar	Radar annotations
Cityscapes [9]	25000	-	●	○	○	○
KITTI [10]	15000	22	●	●	○	○
H3D [11]	28000	160	●	●	○	○
KAIST [12]	-	-	●	●	○	○
RobotCar [13, 14]	-	32	●	●	●	○
SCORP [15]	4000	11	●	○	●	●
ASTYX [16]	500	-	●	●	●	●
DENSE [4]	13500	-	●	●	●	●
nuScenes [17]	34000	1000	●	●	●	●

As described in Table 2.1, the SCORP, ASTYX, DENSE and nuScenes datasets are currently the only publicly available datasets containing annotated radar data. However, the SCORP dataset does not provide the radar data in a point cloud data format, but as raw radar signals. The ASTYX and DENSE datasets, on the other hand, do not provide information about the scene affiliation, which is mandatory to utilize the temporal domain. Therefore, and due to the fact that the nuScenes dataset contains more than two times as many samples as comparable datasets, it is chosen as data basis for this thesis.

2.2 The nuScenes Dataset

The nuTonomy scenes (nuScenes) dataset [17] is a publicly available dataset for autonomous driving that includes annotated radar data. The dataset itself consists of 1000 scenes of 20 s each and includes not only radar data but also lidar, camera and global positioning data as well as information about the ego vehicle motion. The complete sensor setup of the nuScenes data collection platform is shown in Figure 2.1.

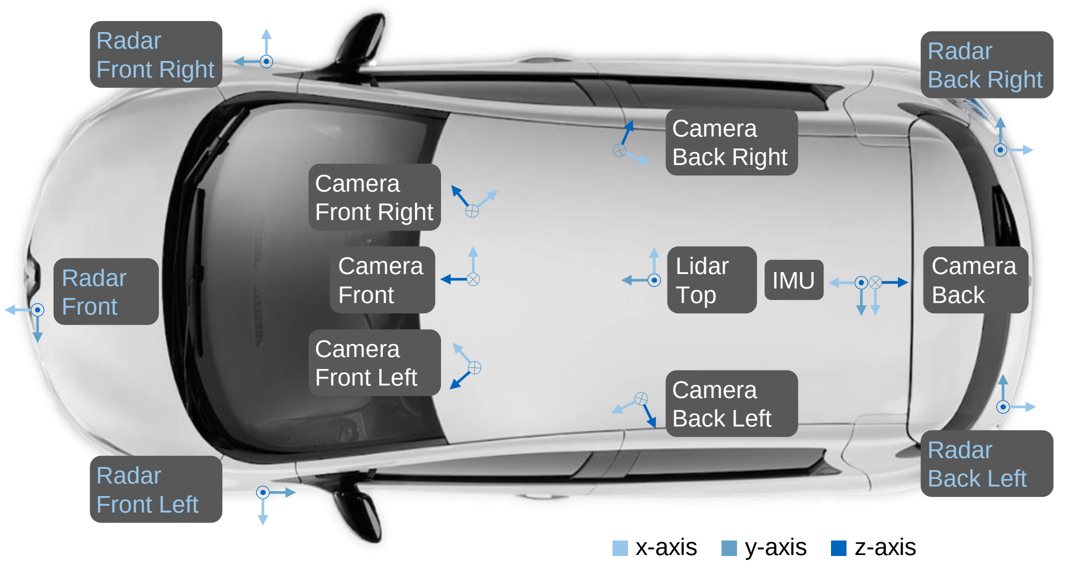


Figure 2.1: Sensor setup of the nuScenes data collection platform according to [17, Fig. 3].

In terms of terminology, a scene denotes a sequence of discrete yet chronologically consecutive keyframes. A keyframe, on the other hand, is defined as an annotated and synchronized aggregation of measurements at a given timestamp. These keyframes are provided at a frequency of 2 Hz as part of a single lidar sweep. All measurements in between of two keyframes are not annotated and denoted as sweeps. These sweeps are provided at the associated sensor capture frequency of approximately 13 Hz for the applied radar sensors.

The deployed radar sensor is the Continental ARS 408-21 Premium, which operates in the 77 GHz frequency band. The sensor itself is connected via the controller area network bus (CAN bus), which limits the number of available radar points to 125 points per sensor and cycle (due to the limited CAN bus bandwidth) [18]. In addition to that, a radar cross section (rcs) filter with a threshold value of -5.0 dBm^2 is applied to the sensor data further limiting its content of information [19]. Any additional specifications of the radar sensors can be found in Appendix A.

Under consideration of the defined limitations and with regard to the sensor specifications, the radar sensor provides eight general and seven quality measurement values per radar point. In addition to these 15 sensor values, the nuScenes dataset also provides the value of the vertical coordinate z, the relative velocity in longitudinal direction compensated by the ego vehicle motion vx_comp and its equivalent in lateral direction vy_comp. A list of all 18 radar channels and their corresponding specifications can be found in Table 2.2.

Table 2.2: List of available radar channels (features) of the nuScenes dataset with their associated value range, resolution and unit. A detailed definition of the radar channels can be found in Appendix A.

Channel ID	Radar Channel	Range	Resolution	Unit
0	x	[-500.0, 1138.2]	0.2	m
1	y	[-258.42, 258.42]	0.2	m
2	z	n.d.	n.d.	m
3	dyn_prob	[0, 7]	1	-
4	id	[0, 255]	1	-
5	rcs	[-5.0, 63.5]	0.5	dBm^2
6	vx	[-128.0, 127.75]	0.25	m/s
7	vy	[-64.0, 63.75]	0.25	m/s
8	vx_comp	n.d.	n.d.	m/s
9	vy_comp	n.d.	n.d.	m/s
10	is_quality_valid	[0, 1]	1	-
11	ambig_state	[0, 7]	1	-
12	x_rms	[0, 31]	1	m
13	y_rms	[0, 31]	1	m
14	invalide_state	[0, 31]	1	-
15	pdh0	[0, 7]	1	-
16	vx_rms	[0, 31]	1	m/s
17	vy_rms	[0, 31]	1	m/s

In addition to the sensor data, the nuScenes dataset also provides annotated (ground truth) data for all 34149 keyframes. These annotations are available as three-dimensional bounding boxes defined by their center coordinates, a rotation quaternion, and a spatial extension [17]. In addition to the spatial information, every annotation includes the information about the visibility of the annotated object as well as a category label for one out of 23 different categories. A distribution of all radar points across these 23 categories of the nuScenes dataset is provided in Figure 2.2.

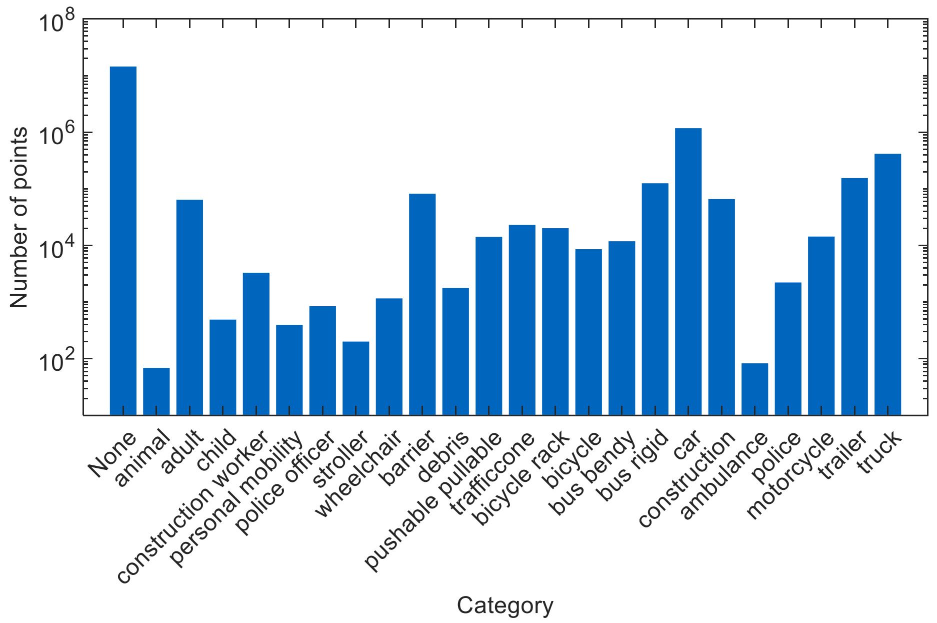


Figure 2.2: Number of radar points per nuScenes category on a logarithmic scale. The None category covers all points without an annotation.

2.3 Data Format

The radar data is provided in form of a point could $\mathcal{P}$, which can be formally defined as a finite set of $\mathrm{n} \in \mathbb{N}$ vectors $\mathbf{p}_i \in \mathbb{R}^{\mathrm{d}}$ with $i = 1, \dots, \mathrm{n}$, whereas n defines the number of radar points of a single keyframe and d denotes the number of dimensions (radar channels) [20, 21]. Provided that the radar point cloud $\mathcal{P}$ originates from an Euclidean space, the points of the finite set $\mathcal{P}$ can be considered as two elements: the point coordinates $\mathcal{C} \in \mathbb{R}^3$ and the point features $\mathcal{F} \in \mathbb{R}^{\mathrm{d}-3}$ [22]. In this case, the point coordinates are considered as structural elements, whereas the point features represent the actual data of the radar point [22].

Given that the radar data is provided as a point cloud of a discrete metric space, the radar data is considered to be:

- unordered,
- invariant under transformations,
- shows interactions among points, with
- entangled feature scales and
- variable densities at different areas [20, 23].

This means that the points $\mathbf{p}_i$ of the point cloud $\mathcal{P}$ are related to each other (interact among each other) and that the properties of a point are not independent but entangled. In addition to the defined characteristics, the non-uniformity of point clouds has to be considered if a sequence of point clouds is taken into account, which means that the number of points is not consistent over time. Besides that, point cloud processing is subject to the general set theory and further described in [24, 25].

To make use of the provided radar data, the point clouds of all keyframes of a scene as well as multiple scenes are mapped to a single input tensor

$$\mathcal{P}_{11} \times \ldots \times \mathcal{P}_{1\mathrm{s}} \times \ldots \times \mathcal{P}_{\mathrm{bs}} \rightarrow \mathbf{X} \in \mathbb{R}^{\mathrm{b \times s \times n \times d}} \tag{2.1}$$

within the model input pipeline. As a result, $\mathbf{X}$ represents the actual input tensor, s the maximum number of keyframes within the batched scenes and b the batch size of the input tensor. At this, d can be interpreted as feature dimension, n as point dimension and s as time dimension, whereas b represents the number of scenes per input tensor.

In addition to the radar data, nuScenes also provides a set of corresponding annotations for every keyframe. To utilize this information the pre-processor generates a set of target values $\mathcal{Y}$ for every point within the input point cloud. This set of target values consists of n vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_\mathrm{n} \in \mathbb{R}^\mathrm{k}$ with k possible classes expressed as a one-hot encoded vector. Just like the input data, the set of target values get mapped to a tensor of target values given by

$$\mathcal{Y}_{11} \times \ldots \times \mathcal{Y}_{1\mathrm{s}} \times \ldots \times \mathcal{Y}_{\mathrm{bs}} \rightarrow \mathbf{Y} \in \mathbb{R}^{\mathrm{b \times s \times n \times k}}. \tag{2.2}$$

A subset of all input and target value pairs $(\mathbf{X}, \mathbf{Y})$ of the nuScenes dataset represents the training data of the supervised machine learning task and builds the basis for all artificial neural networks described in the subsequent section.

2.4 Network Architectures for Point Cloud Data

Building upon this, several methods can be applied to perform a semantic segmentation on the provided radar data. These methods include decision trees, like random forests as used by [26] and [27] as well as logistic regression classifiers. Besides that, support vector machines (SVMs) are often used to classify high dimensional data and are used by [28–31] to segment radar point clouds. Recently, artificial neural networks (ANNs) are widely used since modern network architectures outperform other classification methods, as shown by [27]. For that reason, the following chapter focuses on different ANN architectures to perform a semantic segmentation of the given radar point cloud.

Artificial neural networks, which can be considered as a transfer function between an input $\mathbf{X}$ and an output $\widehat{\mathbf{Y}}$, can be divided into feedforward neural networks and recurrent neural networks (RNNs) [32, p. 166]. Feedforward neural networks, formally defined as

$$\widehat{\mathbf{Y}} = f_\theta(\mathbf{X}) \tag{2.3}$$

are characterized by an unidirectional flow of information [32, p. 168]. This means that the output $\widehat{\mathbf{Y}}$ is not fed back, but only dependent on the current input $\mathbf{X}$ as a result of the transfer function f with parameters θ. If the network includes a feedback connection, the network architecture is referred to as a recurrent neural network and defined as

$$\widehat{\mathbf{Y}}_t = f_\theta(\widehat{\mathbf{Y}}_{t-1}, \mathbf{X}_t), \tag{2.4}$$

where the current output $\widehat{\mathbf{Y}}_t$ is not only dependent on the current input $\mathbf{X}_t$, but also on the output of the previous timestep $\widehat{\mathbf{Y}}_{t-1}$ [32, p. 375]. Both network architectures can be applied to the given segmentation problem and will be discussed in the following.

2.4.1 Feedforward Neural Network Architectures

Feedforward neural networks for point cloud input data can be categorized based on their requirements on the input data and preceding processing steps as well as their satisfaction of different model properties. Network architectures for point cloud data can require a preceding feature engineering step to express the point cloud as a defined feature vector, a preceding data transformation to change the data representation or a preceding data sampling to ensure a fixed input size. On top of this, some network architectures are more suitable to capture spatial relationships among points or have the ability to share kernel weights between multiple points. Based on these aspects, state of the art feedforward neural network architectures for point cloud input data can be grouped into five categories. These networks categories are represented in Table 2.3 and described in the following.

Table 2.3: Requirements and properties of different feedforward neural network architectures for point cloud data. A filled circle represents the satisfaction or demand of a property or requirement, whereas an empty circle represents the opposite of it. A partly filled circle represents the degree to which a property is satisfied in relation to the other network architectures.

Network architecture	Requires feature engineering	Requires a fixed input size	Requires a data transformation	Spatial relationship	Shared kernel
Feature-Based	●	●	●	◔	●
Projection-Based	○	●	●	◕	●
GNN	○	○	●	●	●
Pointwise MLP	○	○	○	●	◑
Point convolution	○	○	○	●	●

Feature Engineering-Based Networks

Feature engineering-based networks rely on an intermediate data representation of the given point cloud by a handcrafted (engineered) feature vector. In comparison to other network architectures, this feature vector is not subject to the spatial domain but represents the point cloud by a defined number of characteristic values. Even if additional features can be used by any of the listed network architectures, feature engineering-based networks solely rely on this intermediate data representation.

Formally, feature engineering-based networks depend on a data transformation given by a mathematical map function

$$f\colon \mathbf{X} \rightarrow \acute{\mathbf{X}}, \tag{2.5}$$

where the original input $\mathbf{X}$ is mapped onto a tuple of defined features $\acute{\mathbf{X}}$ [33, p. 3]. Making use of this technique, feature engineering-based networks map the unordered set of points onto an intermediate data representation of a fixed size. This feature vector represents spatial relationships of the points as well as their unique properties.

Applying this network architecture, SCHUMANN et al. [27] compares the performance of different machine learning approaches based on several engineered features including extreme values, values of the central tendency and the dispersion as well as the eigenvalues of the covariance matrix of the point coordinates. Building upon this, SCHEINER et al. [34] implements a segmentation network for a multiclass road user classification problem and evaluates the impact of 50 different features on the model's performance by using a backward elimination as well as k-fold cross-validation method.

Projection-Based Networks

Projection-based networks are considered to be network architectures able to process point cloud data expressed by a projection of the points onto a discretized representation of the space. This intermediate data representation of a fixed size enables the model to take advantage of conventional network architectures as used by image processing applications.

The projection of the point cloud is defined by a mathematical map function, mapping the unordered set of points onto a grid like structure with a fixed size. This grid like structure is a discretized representation of the underlying space and can be expressed by either a two-dimensional matrix, a three-dimensional voxel structure or any other fixed-size structure of arbitrary dimensionality. Based on this projection several methods can be applied to process the given radar data.

LOMBACHER et al. [35] used a convolutional neural network (CNN) to perform a semantic segmentation on a two-dimensional projection of the radar data on an occupancy grid. Furthermore, state of the art results are achieved by utilizing a three-dimensional projection-based network architecture for the semantic segmentation of light detection an ranging (lidar) sensor data [36]. Besides that, sparse convolutional neural networks [37] as well as multi-view approaches [38] are used to counteract the demand of high dimensional projection-based networks on system resources – further described in [37].

Graph Neural Networks

Graph neural networks (GNNs) are able to process a graph representation of data originating from a non-Euclidean space. This ability enables GNNs to operate on an unordered set of points with varying input sizes, which is expressed as a graph. Therefore, a preceding data transformation is required to utilize a GNN for radar point cloud processing.

This intermediate data structure represents the radar point cloud as a pair of two sets. The first set consist of the radar points themselves and represents the graph vertices, whereas the second set defines the graph edges and represents the point relationships [39]. Both, the vertices and the edges, can be associated with a dedicated weight variable to express either point features or relationship values, such as the distance for example. This graph representation of the radar point cloud can then be processed by an artificial neural network, further described in [39]. However, this type of network architectures has not yet been applied to automotive radar data related problems and is open for research.

Pointwise Multilayer Perceptron Networks

In comparison to previously discussed network architectures, pointwise multilayer perceptron (MLP) networks operate directly on point cloud input data. For that reason, pointwise MLP networks are able to process unordered sets of points with varying input sizes without relying on an intermediate data representation. This property enables the utilization of pointwise MLP networks for semantic segmentation tasks directly on point cloud input data.

This ability is a result of the network design, which models the relationship between the elements $\mathbf{p}_i$ of a finite set $\mathcal{P}$ as a composite of two functions $f \circ g$ [40]. The response $\tilde{\mathrm{r}}$ of the composite

$$\tilde{\mathrm{r}} = f_\theta\left(g_\psi(\mathbf{p}_1), \dots, g_\psi(\mathbf{p}_\mathrm{n})\right), \tag{2.6}$$

with parameters θ and ψ, represents the relationship between all elements of the set and can be interpreted as a sample (or scene) representation of the given point cloud $\mathcal{P}$ [20]. The kernel function g operates on the individual elements and can be implemented as an elementwise convolution operation. The aggregation function f aggregates the individual activations to receive a global representation of the given point cloud. This fundamental operation of pointwise MLP networks is able to approximate any arbitrary complex continuous function [20] and is illustrated in Figure 2.3.

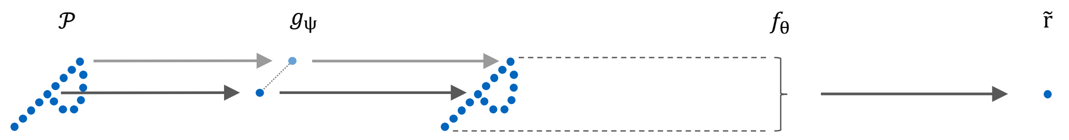


Figure 2.3: Visualization of the core functionality of a pointwise MLP network to obtain a global representation $\tilde{\mathrm{r}}$ of the given point cloud $\mathcal{P}$ by applying a composite of two functions $f \circ g$. The shared kernel function g is applied to every point of the point cloud $\mathcal{P}$ to receive a pointwise activation which is aggregated by f to compute a global feature vector $\tilde{\mathrm{r}}$.

PointNet [20] applies this technique to compute a global feature vector from a given set of points to perform either a classification or semantic segmentation on the given point cloud. The main idea of the network is the utilization of a single symmetric function f to encode the global feature vector $\tilde{\mathrm{r}}$ as the maximum response among all points [22]. Therefore, the kernel function g is implemented as a shared MLP, which acts as a set of learned spatial encodings, and the aggregation function f is implemented as max-pooling across all elements [20]. Due to this model structure, PointNet is invariant to input permutations and can handle input data with varying input sizes, but lacks the ability to consider local spatial relationships between the data points [23].

To overcome these limitations PointNet++ [23] was designed in a hierarchical way to capture local structures within the input point cloud. This hierarchical encoding is achieved by sequentially applying three major processing steps. First, a defined number of center points is sampled from the given point cloud according to an iterative farthest point sampling method [23]. Second, a maximum number of points in the local neighborhood of each center point is determined by a ball query method to form local groups [23]. Finally, a PointNet is applied to every group to encode the local features of the centroid's neighborhood [23]. This sequence of operations is denoted as set abstraction (SA) and can be applied in a hierarchical way to obtain a global representation of the input point cloud, as shown in Figure 2.4.

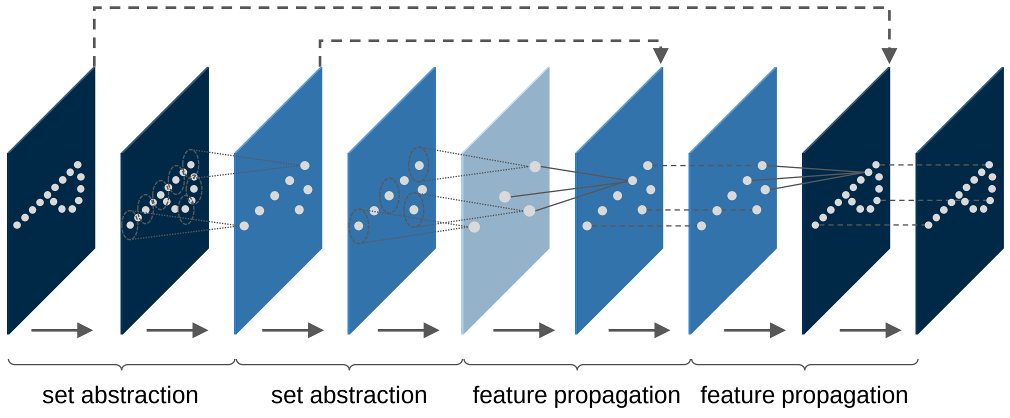


Figure 2.4: Illustration of the set abstraction and feature propagation module of the PointNet++ architecture in accordance with [23].

The restoration of the original point cloud – the decoding of the global feature vectors – is accomplished by an equal number of feature propagation (FP) modules. Within an FP module the points of the local neighborhoods get associated with the encoded features of the corresponding center points by a spatial interpolation method. This way, the original point cloud is restored in a hierarchical way, as illustrated in Figure 2.4.

Applying this network architecture, DANZER et al. [41] used two consecutive PointNet models to perform a semantic segmentation on a 2D radar point cloud and estimates bonding boxes to solve a binary classification problem. On this basis, SCHUMANN et al. [21] implemented a PointNet++ architecture to train a semantic segmentation model for a multi-class segmentation problem. Recently, CENNAMO et al. [42] proposed a network architecture with multiple PointNet++ networks in combination with an attention mechanism to perform a semantic segmentation on automotive radar data.

Point Convolution Networks

Point convolution networks apply a kernel-based convolution operation directly to the input point cloud. Unlike pointwise MLP networks, the convolution kernel is not limited to a single point but can be applied to a set of points. Moreover, point convolution networks are able to represent spatial relationships by a shared kernel and can operate on unordered sets of points with varying input sizes.

Convolutional neural networks – including point convolution networks – are based on a discrete convolution operation of an input function with a kernel function to compute the desired output[32, pp. 331-334]. Applying this method to an unordered set of points, the discrete convolution of a set of point features $\mathcal{F} \in \mathbb{R}^{\mathrm{d}-3}$ with a kernel function g at a point $\mathbf{x} \in \mathbb{R}^3$ is defined as

$$(\mathcal{F} * g)(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{N}_\mathbf{x}} g(\mathbf{x}_i - \mathbf{x})\mathbf{f}_i, \tag{2.7}$$

where $\mathbf{x}_i \in \mathbb{R}^3$ represents the point coordinates to the corresponding point features $\mathbf{f}_i \in \mathbb{R}^{\mathrm{d}-3}$ of the finite set $\mathcal{P}$, whereas $\mathcal{N}_\mathrm{x} \subseteq \mathcal{P}$ denotes a set of points in the neighborhood of $\mathbf{x}$ [22]. Under consideration of this convolution operation it is the objective of the model training to learn a suitable kernel function by adjusting the kernel weights.

Based on this operation, XU et al. [43] developed a kernel function with different weights for each neighboring point depending on the neighbor's distance-wise order, which leads to a spatially inconsistent filter kernel [22]. To overcome this problem, Flex-Convolution [44] is built on a spatially independent filter kernel, which applies a linear kernel function to the k-nearest-neighbors. In contrast, LI et al. [45] proposed a kernel function with kernel-point-based filter weights in association with a correlation function to realize a discrete convolution operation on point cloud data.

On this basis, KPConv [22] introduces a kernel point convolution (KPConv) based on a set of kernel points with a spatial location and an associated kernel weight. Therefore, the kernel function

$$g(\bar{\mathbf{x}}_i) = \sum_{j<\mathrm{n_k}} h(\bar{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)\mathbf{w}_j \tag{2.8}$$

describes the influence of the kernel weights $\mathbf{w}_j$ onto the neighboring points $\mathbf{x}_i$ by a correlation function h, where $\bar{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}$ is the neighbor's position centered on $\mathbf{x}$ and $\tilde{\mathbf{x}}_j \in \{\tilde{\mathbf{x}} \mid \mathrm{j} < \mathrm{n_k}\} \subset \mathcal{B}_\mathrm{r}$ denotes the set of $\mathrm{n_k}$ kernel points within the neighborhood domain $\mathcal{B}_\mathrm{r} = \{\bar{\mathbf{x}} \in \mathbb{R} \mid \|\bar{\mathbf{x}}\| \leq \mathrm{r}\}$ with radius r [22]. The correlation function h, which defines the degree to which a kernel point is associated with the neighboring points, is defined as

$$h(\bar{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = max\left(0, 1 - \frac{\|\bar{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|}{\sigma}\right), \tag{2.9}$$

where σ is the area of influence of the kernel points and will be chosen according to the input density [22]. A graphical representation of the described kernel point convolution is provided in Figure 2.5.

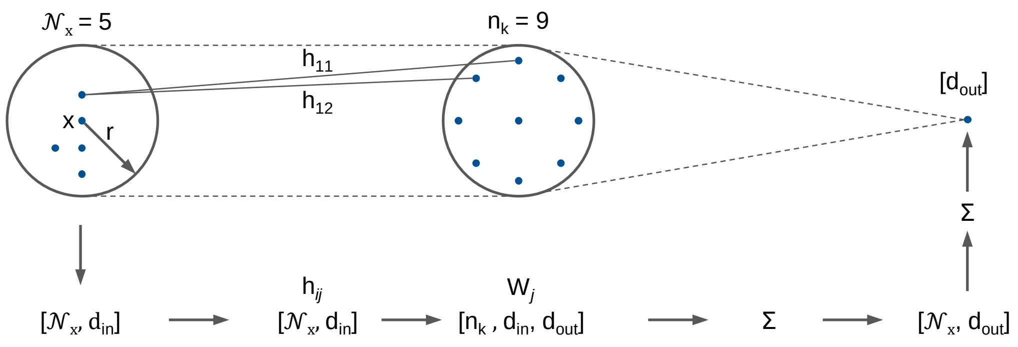


Figure 2.5: Graphical representation of the kernel point convolution in association with [22].

2.4.2 Recurrent Neural Network Architectures

Recurrent neural networks enhance the capabilities of feedforward neural networks by the ability to consider neuron outputs of preceding timesteps [32, pp. 373-374]. The introduction of this feedback loop enables RNN architectures to utilize temporal information of input data sequences. However, due to the given data format (as described in section 2.3), recurrent neural networks for point cloud data must be able to process unordered sets of points with varying input sizes over time.

To process point cloud data recurrently, an association function is required to associate the points of the previous timestep with the points of the current timestep [46]. Due to the unordered data structure, this association relies on the point information itself rather than the point index within the data structure and can be implemented as one of the methods discussed in subsection 2.4.1.

Based on a pointwise MLP approach, SCHUMANN et al. [1] used a modified PointNet++ architecture to associate the radar point cloud of the previous timestep with the current input point cloud. To achieve this, a new layer is introduced which calculates an additional feature vector for every point in the current input point cloud based on the previous point cloud. This layer combines both the current and the previous point cloud to apply a modified PointNet++ operation on the cumulated points. This modified operation uses the new points as center points to form local neighborhoods from the previous point cloud only. The neighborhood points are then encoded by a PointNet layer to receive an additional feature vector for every center point. Doing so, the information of the previous timestep gets encoded into the current input point cloud. However, simple feedback loops – like this one – often encounter vanishing or exploding gradients, as shown by [46, 47].

To overcome this problem, FAN and YANG [46] proposed an RNN architecture for point cloud input data based on the long short-term memory (LSTM) cell of HOCHREITER and SCHMIDHUBER [47]. The LSTM cell itself consists of two states (the hidden state $\mathbf{H}$ and the cell state $\mathbf{C}$) and three internal gates to regulate the flow of information. The input gate $\mathbf{I}$ controls the flow of information into the cell, the forget gate $\mathbf{F}$ regulates the remaining information within the cell and the output gate $\mathbf{O}$ controls the transmission of information. To adapt this cell design for point cloud data processing several modifications are made in order to achieve a model architecture invariant to input data permutations and with the ability to handle input data of varying sizes.

First, the gate units of the LSTM cell, which are originally implemented as fully connected layers, are replaced by a modified PointNet++ layer p_θ, similar to [1]. Therefore, the points of current input point cloud $\mathbf{X}_t$ are used as center points to form local neighoorhoods from the points of the previous timestep $\mathbf{H}_{t-1}$ and encode their information into the current input points by applying a PointNet operation. Second, another modified PointNet++ layer is applied to the previous cell state $\mathbf{C}_{\mathrm{t-1}}$ to associate the points of the previous timestep with the points of the current input point cloud. This new step of calculating an intermediate cell state $\hat{\mathbf{C}}_{\mathrm{t-1}}$ is required since the number of points can vary over time and the order of the points is not preserved over timesteps. All changes to the original LSTM architecture are represented in Figure 2.6 and highlighted in blue. Nevertheless, PointRNN has not yet been applied to radar data related problems nor has any other LSTM-based architecture been used to semantically segment automotive radar data raising the demand on investigating the utilization of an LSTM-based model architecture.

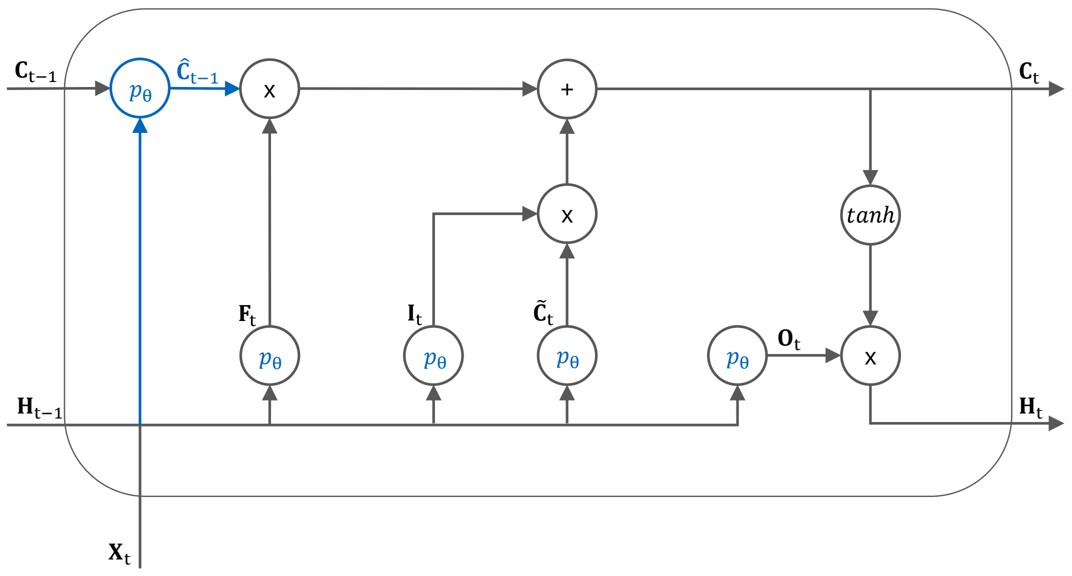


Figure 2.6: Computational graph of the PointLSTM [46] cell with input $\mathbf{X}$, hidden state $\mathbf{H}$ and cell state $\mathbf{C}$ of the timestep t.

3 Model Development

Based on the current state of the art and in line with the method defined in section 1.2, the model development follows a data driven development strategy. According to this strategy, the process chain is split into dedicated modules, represented in Figure 3.1 and described in the following.

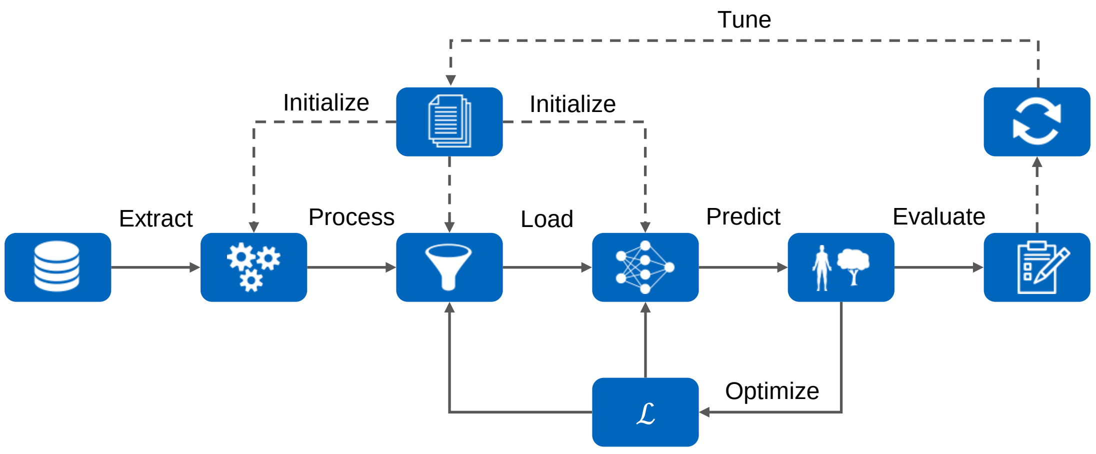


Figure 3.1: Overview of the model pipeline represented by three major parts. The center part represents the model's forward pass to create a prediction from a given dataset, the inner loop (on the bottom) represents the model training by a gradient-based optimization process and the outer loop (at the top) visualizes the hyperparameter tuning by adjusting the initial configuration.

The process pipeline originates from the chosen dataset and processes the given data in a preceding pre-processing step, as discussed in section 3.1. The pre-processed dataset is then fed to the input pipeline – described in section 3.2 – and loaded onto the model. Based on this data, the network architecture is defined in section 3.3 and trained to predict a class label for every point in the given input point cloud.

The model training describes the process of iteratively adjusting the model weights to minimize the value of the loss function, which is defined in section 3.4. This optimization process is realized by the optimizer, which is defined in section 3.5. The goal of this model training is to obtain an optimal prediction result measured by the defined evaluation metric of section 3.6. To reach this goal several experiments are executed to explore the parameter space and tune the performance of the model based on the experimental design, explained in section 3.7.

3.1 Data Pre-Processing

The data pre-processor builds the link between the dataset and the model input pipeline. Therefore, the pre-processor module is not independent from the individual datasets but built upon their individual data structure. Under consideration of this dependency, the preceding pre-processor module has three main purposes:

1. the provision of the model input data,
2. the determination of the target values and
3. the homogenization of the data format.

The provision of the input data describes not only the extraction of the point cloud information but realizes the user configuration. This configuration includes the aggregation, filtering, and mapping of the point cloud data. Therefore, the pre-processor combines the information of multiple radar sensors and aggregates the point clouds of multiple radar sweeps. In addition to that, the point cloud is filtered by properties of the scene, the keyframe and the points themselves. Furthermore, the pre-processor allows the mapping of individual radar channels to defined values.

Applying this functionality to the utilized nuScenes [17] dataset, the point cloud information of all five radar sensors is used and three consecutive sweeps are aggregated to obtain the input point cloud. The point cloud is not filtered by any of the radar quality values listed in Table 2.2 and all four recording locations are taken into account. The resulting point cloud data is limited to eight radar channels, to decrease the demand on system resources and improve the model performance, as discussed in [48, pp. 29-34]. The remaining radar channels of the input point cloud are the three coordinate values, the dynamic property, the rcs value, the pdh0 value as well as the compensated relative velocity in longitudinal and lateral direction.

The determination of the target values describes the process of assigning a class label to every point of the obtained point cloud. The label is determined based on the spatial location of the points and assigned if the point is located within a provided bounding box (annotation). Therefore, the bounding box dimensions are increased by a configurable factor (denoted as whl_factor) to take positional tolerances into account. All points that are not belonging to a given annotation are considered as background and labeled as None class.

The actual label of all points within a bounding box depends on the annotation category and the annotation attribute. The assignment process allows the remapping of nuScenes categories or annotation attributes to user configured classes. This allows the combination of multiple categories to a single class and enables the ability to assign class labels based on annotation attributes rather than on annotation categories. Furthermore, the assignment to the None class and even the removal of the overall background is possible.

Based on this target value determination, three different dataset compositions are created from the nuScenes dataset. The first composition, which serves as benchmark dataset, is created according to the results of [48] and represented in Table 3.1. The second composition focuses on the differentiation of pedestrians from the surrounding points and represents a binary segmentation problem. The third composition is based on the nuScenes annotation attributes rather than on annotation categories and is made up of two classes, one representing all points belonging to the moving vehicle attribute and another combining all other points. In the following, these dataset compositions are denoted as benchmark, pedestrian, and dynamic vehicle dataset.

Table 3.1: Proposed dataset compositions as combination of multiple nuScenes categories to user configured classes.

nuScenes category	Benchmark dataset	Pedestrian dataset
None	None	None
animal	Pedestrian	None
human.pedestrian.adult	Pedestrian	Pedestrian
human.pedestrian.child	Pedestrian	Pedestrian
human.pedestrian.construction_worker	Pedestrian	Pedestrian
human.pedestrian.personal_mobility	Cycle	None
human.pedestrian.police_officer	Pedestrian	Pedestrian
human.pedestrian.stroller	Pedestrian	Pedestrian
human.pedestrian.wheelchair	Pedestrian	Pedestrian
movable_object.barrier	None	None
movable_object.debris	None	None
movable_object.pushable_pullable	None	None
movable_object.trafficcone	None	None
static_object.bicycle_rack	None	None
vehicle.bicycle	Cycle	None
vehicle.bus.bendy	Vehicle	None
vehicle.bus.rigid	Vehicle	None
vehicle.car	Vehicle	None
vehicle.construction	Vehicle	None
vehicle.emergency.ambulance	Vehicle	None
vehicle.emergency.police	Vehicle	None
vehicle.motorcycle	Cycle	None
vehicle.trailer	Vehicle	None
vehicle.truck	Vehicle	None

In a third step, the data of all keyframes of a scene gets combined and transformed to a homogenous data format. Therefore, the point cloud data (.pcd) is serialized to a cross-platform, cross-language compatible and efficient protocol buffer message (.tfrecord), which is recommended to use in combination with a TensorFlow-based input pipeline [49].

3.2 Input Pipeline

The input pipeline is responsible for supplying the pre-processed data to the model and is split into three stages:

1. the extraction of the data,
2. the transformation of the data format and
3. the loading of the data onto the accelerator device [48].

In the first stage, the serialized protocol buffer messages are extracted from the files and passed to a deserializer. The deserializer parses the extracted messages according to a defined protocol to receive the actual point cloud data $\mathcal{P}$ and their corresponding target values $\mathcal{Y}$. In the second stage, the received data is randomly shuffled and transformed into an input tensor $\mathbf{X}$ as well as a corresponding tensor of one-hot encoded target values $\mathbf{Y}$. To transform the point cloud data and batch the received tensors a zero-padding method is applied. This method describes the process of adding zero-elements to the data until the length of all elements in the batch is equal within every dimension. Finally, the dataset is cached, prefetched and loaded onto the accelerator device (GPU) to be retrieved by the model training.

3.3 Network Architecture

The network architecture defines the transfer function between the input $\mathbf{X}$ and the output $\hat{\mathbf{Y}}$ to predict a class label for every point within the input point cloud. Under consideration of the provided data format (described in section 2.3) and with regards to the raised research questions in section 1.2, the following requirements can be imposed on the network design.

- Order invariance: a model consuming n points must be invariant to $\mathrm{n}!$ permutations of the input set [23].
- Information intactness: the number of elements within the input and output set must be equal without any loss of information [50].
- Interaction among points: the model should be able to capture relationships among neighboring points [23].
- Location variance: the model should be able to capture dynamically changing relations among points over time [51].
- Robust to transformations: the model should be robust to correlation-preserving transformations of the input point cloud, such as scaling or rotating [23].

Besides the requirements on the network architecture itself, the modularity and flexibility of the model structure is a key objective of the model development to realize a simple evaluation of different network architectures.

To achieve this, a modular development platform, as characterized by the macroscopic model structure represented in Figure 3.2, is proposed. This structure consists of five model placeholders each dedicated to a specific task. The input model is dedicated to data related transformations and is a key element for the realization of a network architecture robust to transformations. The encoder model encodes the provided input data to extract the essential information from the given data and the bridge model can be used to further process this data. The decoder model restores the original data based on the encoded information and the given input data. Finally, the output model maps the decoded information onto the class labels to realize the semantic segmentation of the input point cloud. In terms of terminology, each submodel is split into multiple modules containing the actual layers of the network which realizes the network operations – explained in the following.

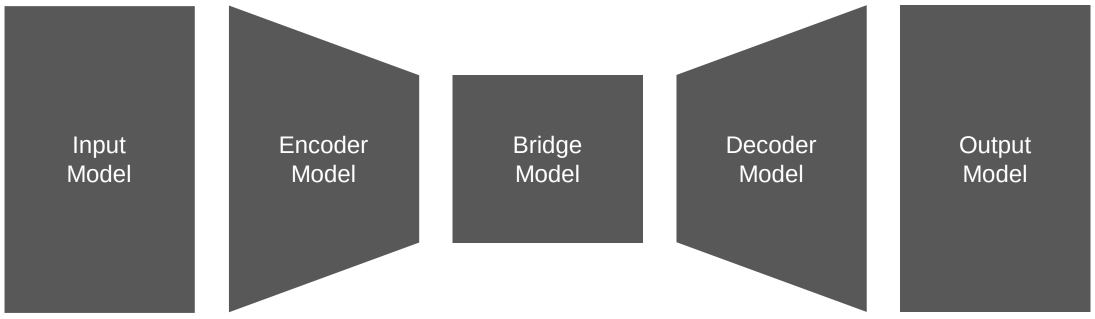


Figure 3.2: Macroscopic model structure as development platform with five configurable submodels.

In consideration of the above defined model structure and to fulfill the model requirements, a feedforward neural network is derived from the current state of the art in subsection 3.3.1 before a feedback loop is introduced in subsection 3.3.2 to utilize the temporal domain.

3.3.1 Feedforward Neural Network Architecture

The feedforward neural network architecture is divided into the submodels discussed above without the utilization of the bridge model and realizes the forward pass of the model. The model does not consider the temporal information of the input data, which means that the requirement on the location variance cannot be met but is subject to the recurrent neural network architecture described in subsection 3.3.2. The four remaining submodels of the feedforward neural network architecture are described in the following.

First, the input model normalizes the given data to improve the stability of the training and to build the basis for a model robust to transformations [52, pp. 29-35]. This input data normalization is implemented as a per channel min-max-scaling, where the input value $\mathbf{X}_{ijl}$ is scaled to an interval of $[0,1]$ according to

$$\mathbf{X}'_{ijl} = \frac{\mathbf{X}_{ijl} - \min}{\max - \min}, \tag{3.1}$$

with a minimum value $\min$ and maximum value $\max$ of the channel l. The minimum and maximum values of the individual channels are chosen in accordance with the channel's value range defined in Table 2.2, except for the point coordinate channels. The scaling parameters of the point coordinate values are determined based on the utilized dataset and defined as ±250 m for the x- and y-coordinate values as well as zero and one meter for the z-coordinate values, further explained in [48].

The encoder model – which builds the core of the feedforward neural network – is derived from the KPConv [22] architecture and based on the kernel point convolution operation discussed in subsection 2.4.1. The selection of this network architecture over others is motivated by its suitability for point cloud input data and its ability to represent point relationships by a shared convolution kernel, as shown in Table 2.3. Given that feature engineering-based networks as well as projection-based networks rely on an input data format of a fixed size, the information intactness can only be ensured if the discretization of the space is chosen in accordance with the sensor resolution. However, such a fine discretization of the space would lead to a high demand on system resources and a sparse data representation, making it impractical for real world applications [21]. On the other hand, GNNs like [53–55] are based on an approach very similar to the utilized kernel point convolution, but require a data transformation to express the given point cloud as a graph, hance increasing the overall model complexity. In addition to that, GNNs tend to focus more on edge relationships rather than on points relative positions and therefore tend to neglect the arrangement of the points [22]. In comparison to that, pointwise MLP networks can operate directly on point cloud input data but are limited to the utilization of shared MLP layers, making the convolution operation more complex and increasing the difficulty of training convergence [22]. For those reasons, the KPConv [22] architecture is chosen as core model architecture and described in the following.

The encoder architecture is built of three set abstraction modules in association with the network architecture of [21, 48]. The bottleneck structure of this submodel is created by a hierarchical implementation of the SA modules to enforce an element reduction and to extract the essential information of the given point cloud. To achieve this, the three-stage encoder implementation is initialized according to the parameters listed in Table 3.2, where the number of center and neighboring points is reduced while the radius of the local neighborhoods is increased. At the same time, the number of filters is increased to encode the information of the overall point cloud into a reduced number of center points.

Table 3.2: Key hyperparameter settings of the three encoder modules of the proposed network architecture.

Encoder module	Number of center points	Max. number of neighboring points	Neighborhood radius r_0	Number of filters
0	512	16	0.016	16, 16, 32
1	128	8	0.032	32, 32, 64
2	16	4	0.064	64, 64, 128

The SA modules are implemented as residual network (ResNet) blocks [56] according to [22] and initialized as discussed above. Each of these ResNet blocks consists of a set of three convolution layers and a skiplink connection, as represented in Figure 3.3. The core of this network block is implemented as a KPConv layer, as described in subsection 2.4.1, which is surrounded by two shared MLP layers. The actual set abstraction is executed by the KPConv layer, which first samples a defined number of center points to build local neighborhoods before a kernel point convolution is applied to encode the information of the neighborhoods within the center points. To associate the skiplink data with the encoded point cloud, the same iterative farthest point sampling method is applied to the skiplink data before the data is processed by a shared MLP layer to align the number of point features. Finally, the encoded data and the skiplink data are added up and passed to an activation function before the data is handed over to the next module.

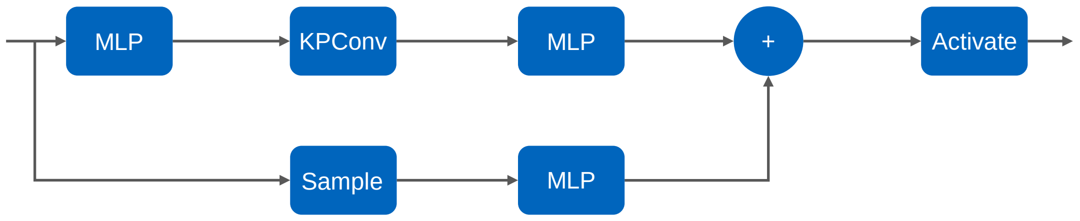


Figure 3.3: ResNet block of the KPConv architecture according to [22]. The sample layer is implemented as an iterative farthest point sampling and the activation function is user configurable but for our purposes defined as rectified linear unit.

The decoder model is chosen in accordance with the PointNet++ [23] architecture and implemented in association with [21]. This decision on the network design is based on two considerations. First, the comparability between different network architecture as well as the current state of the art should be preserved by changing as few parameters as possible. Second, the utilization of more complex decoder structures does not significantly enhance the segmentation quality of the network, as shown by [22]. Therefore, the decoder is implemented by three FP modules, as described in subsection 2.4.1.

The output model structure follows a similar approach and uses a number of three shared MLP layers in combination with two dropout layers, as proposed by [21]. This model design allows a continuous reduction of the feature dimensions to determine a k-dimensional class score for every point, while keeping the number of model parameters low. This characteristic and the additional dropout layers are important to prevent the model from overfitting, as shown by [57].

Based on this network architecture, an end-to-end point cloud segmentation model can be implemented and is evaluated in the following. However, this network architecture lacks the ability to consider temporal information of consecutive keyframes, which is why an extension of the feedforward neural network architecture is required.

3.3.2 Recurrent Neural Network Architecture

The extension of the network architecture, by introducing a feedback loop to utilize the temporal domain, can be realized on different model levels. On the one hand the RNN can be implemented within the bridge model to use the advantage of a known and fixed input data size or on the other hand within the encoder model to utilize the temporal information on different encoding levels. While currently available RNN architectures can only be used within the first approach, the second one could benefit from the available information on the different encoding levels.

Nevertheless, both approaches require a combination of RNN architectures and non-recurrent neural network architectures within the same model. This is challenging, because RNN architectures are based on input data with an additional time dimension which is not required for non-recurrent neural networks. However, this problem can be solved with three different approaches. First, the individual timesteps can be processed in sequence, while storing the internal state of the RNN layers and use it to initialize the next processing step. However, this approach would require equally long sequences across multiple scenes to be able to parallelize the model training (by batching the input data) without losing input data by cutting of scenes at a specified length. The second approach would require an extension of all non-recurrent layers by an artificial dummy dimension across which the layer operations can be shared. However, this would require a high implementation effort and increase the overall model complexity. Besides those two approaches, the non-recurrent neural network layers can be unrolled along the time dimension so that every timestep is processed by an individual layer instance before they get combined again before the RNN layer. This approach (which is called time distribution) can be implemented very efficiently by reshaping the input tensors and parallelizing the network operations during the model training, which is why it is chosen as preferred strategy.

Following this technique, two network architectures with a recurrent bridge model are defined in a first step. These network architectures include a model with a standard LSTM layer according to [47] within the bridge model and one with a convolutional LSTM layer according to [58]. The main difference between these two models is that the first one is based on fully connected layers within the LSTM cell while the second one uses a convolution operation to realize the LSTM gates and layers [58]. In addition to that, the LSTM layer supports just three-dimensional input data so that the point and feature dimension have to be combined (flattened).

To overcome these limitations and to utilize the temporal information across multiple encoding levels a hierarchical RNN architecture for point cloud input data is developed. To achieve this, an LSTM cell invariant to input permutations and with the ability to process input data with changing input sizes across timesteps is developed. This novel network architecture is called kernel point convolutional long short-term memory (KPLSTM) and is derived from the PointLSTM cell described in subsection 2.4.2 and illustrated in Figure 2.6.

The KPLSTM cell is developed by replacing of the PointNet layers p_θ within the original PointLSTM cell. These PointNet layers are replaced by a modified KPConv layer to overcome the limitations of the PointNet architecture on the kernel size. The modification of the KPConv layer is inspired by [1] and concerns the encoding method of the layer. Thus, the modified KPConv layer receives not just a single input point cloud but both the current input point cloud $\mathbf{X}_t$ as well as the point cloud of the previous hidden state $\mathbf{H}_{t-1}$. Within this new layer, the current input points are used as center points while the previous point cloud is used as neighbor points and associated to the current input points by applying a KPConv kernel to the local neighborhood. Therefore, the information of the previous timestep gets encoded into the current input and the three gate units of the proposed KPLSTM cell can be defined as

$$\mathbf{I}_t = \sigma\big(KPConv(\mathbf{X}_t, \mathbf{H}_{t-1})\big), \tag{3.2}$$

$$\mathbf{F}_t = \sigma\big(KPConv(\mathbf{X}_t, \mathbf{H}_{t-1})\big), \tag{3.3}$$

$$\mathbf{O}_t = \sigma\big(KPConv(\mathbf{X}_t, \mathbf{H}_{t-1})\big), \tag{3.4}$$

where σ denotes a sigmoid activation function. Following the same idea, the intermediate cell state $\hat{\mathbf{C}}_{t-1}$ is determined based on the current point coordinates $\mathbf{X}'_t$ (not considering the current point features) and the internal cell state $\tilde{\mathbf{C}}_t$ is calculated according to the gate units but with a hyperbolic tangent activation function, given as

$$\hat{\mathbf{C}}_{t-1} = KPConv(\mathbf{X}'_t, \mathbf{C}_{t-1}), \tag{3.5}$$

$$\tilde{\mathbf{C}}_t = tanh\big(KPConv(\mathbf{X}_t, \mathbf{H}_{t-1})\big). \tag{3.6}$$

Based on these operations, the new cell state $\mathbf{C}_t$ and the new hidden state $\mathbf{H}_t$ can be calculated according to the standard LSTM implementation by HOCHREITER [47] with the elementwise Hadamard product $\odot$, as

$$\mathbf{C}_t = \mathbf{F}_t \odot \hat{\mathbf{C}}_{t-1} + \mathbf{I}_t \odot \tilde{\boldsymbol{C}}_t, \tag{3.7}$$

$$\mathbf{H}_t = \mathbf{O}_t \odot tanh(\mathbf{C}_t). \tag{3.8}$$

Making use of this technique, the novel set abstraction module is defined by the combination of a KPLSTM layer and two KPConv layers, which allows a comparable implementation to [22, 23]. The network architecture is initialized with the same hyperparameter settings, as represented in Table 3.2, and a graphical representation of a SA model is given in Figure 3.4.

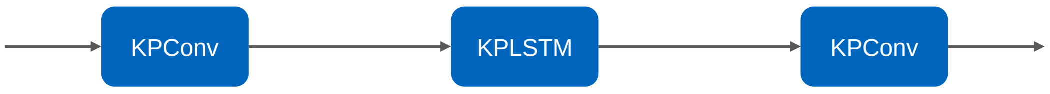


Figure 3.4: Graphical representation of the set abstraction module of the KPLSTM architecture.

3.4 Loss Function

The loss function is the subject of the gradient-based optimization process that aims to find the "best available" set of model parameters to minimize the loss value [59, p. 41]. This optimization process is referred to as the model training, whereas the loss function evaluates the model's prediction $\widehat{\mathbf{Y}}$ in relation to the corresponding target values $\mathbf{Y}$. The goal of this training is the optimization of the model's performance, with the loss function representing the objective function of this optimization process [32, p. 275].

To achieve this goal the loss function design has be chosen with respect to the underlying data distribution and with regards to the aspired project goal [32, p. 275]. Therefore, a loss function according to [48] is chosen, which is designed for model trainings on imbalanced datasets and multi-class classification problems. The loss value of a single keyframe is defined as

$$\begin{aligned}\mathcal{L}(\mathbf{Y},\widehat{\mathbf{Y}}) = -\mathrm{a}\sum_{i=0}^{\mathrm{n}} \mathbf{c}\,\mathbf{Y}_{i:} \sum_{j=0}^{\mathrm{k}} \mathbf{Y}_{ij}\alpha_j\left(1-\widehat{\mathbf{Y}}_{ij}\right)^{\gamma} log\left(f\left(\widehat{\mathbf{Y}}_{ij}\right)\right) \\ + \left(1-\mathbf{Y}_{ij}\right)\left(1-\alpha_j\right)\widehat{\mathbf{Y}}_{ij}^{\gamma}\, log\left(1-f\left(\widehat{\mathbf{Y}}_{ij}\right)\right)\end{aligned} \tag{3.9}$$

and is based on the loss function of Lin et al. [60]. To calculate the overall loss value of a training step the loss values of all keyframes and scenes within a batch are aggregated (summed up). The loss function itself consists of six major components, each described in the subsequent paragraphs, and can be split into a true part for $\mathbf{Y}_{ij} = 1$ and false part for $\mathbf{Y}_{ij} = 0$.

The first part – the logit function f – maps the output of the model to a suitable value range $f: \widehat{\mathbf{Y}}_{ij} \rightarrow [\varepsilon, 1-\varepsilon]$, where $\varepsilon \ll 1$ represents a small value different from zero to ensure numerical stability. This step, however, is not required if the output activation function satisfies this requirement already. Second, a logarithm function is applied to the prediction to penalize deviations from the target values $\mathbf{Y}_{ij}$, whereas large deviations are penalized more than small deviations. In a third step, a modulating factor – raised to the power of the focal value $\gamma \geq 0$ – is multiplied to reduce the relative loss for well-classified examples and to put more focus on misclassified examples [60]. Next, a class depended balance factor α_j is applied to the loss function to balance between the true and the false part of the equation. This balance factor is introduced to pay more attention to either false positives or false negatives and its influence on the loss value is illustrated in Figure 3.5.

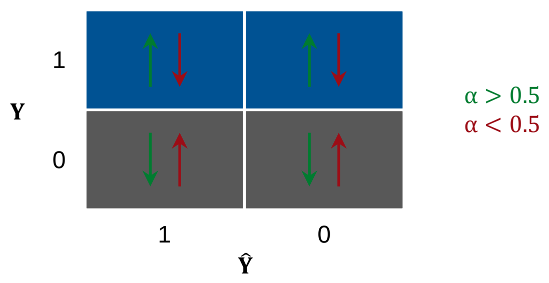


Figure 3.5: Influence of the balance factor α on the loss value, where an up facing arrow indicates an increase of the loss value and an down facing arrow indicates a decreasing loss value – in comparison to an alpha setting of $\alpha = 0.5$.

In a fifth step, a static class weight is applied to the loss value by multiplying the defined vector of class weights $\mathbf{c} \in \mathbb{R}^{\mathrm{k}}$ with the one-hot encoded target value vector $\mathbf{Y}_{i:}$ of the *i*th point (as a slice of the target value tensor). The class weights are determined based on the underlying class distribution of the utilized dataset to compensate for the effects of imbalanced datasets on the model training. Therefore, the class weight of the *j*th class is defined as

$$\mathbf{c}_j = \frac{1}{\left|\mathcal{P}'_j\right|} \frac{|\mathcal{P}|}{\mathrm{k}}, \tag{3.10}$$

where $\left|\mathcal{P}'_j\right|$ denotes the number of elements that belong to class j, for $j = 0, \dots, \mathrm{k}$ and $|\mathcal{P}|$ the number of elements in the finite set $\mathcal{P}$ (the cardinality of $\mathcal{P}$) [61].

As final step, a compensation factor a is deployed to account for the differences in the loss value caused by the applied balance factors. This compensation factor, given by

$$\mathrm{a} = \frac{1}{|\mathcal{P}|} \sum_{j=0}^{\mathrm{k}} \frac{\left|\mathcal{P}'_j\right|}{\alpha_j}, \tag{3.11}$$

is important to ensure comparability between different model trainings and to preserve the magnitude of the loss value. The overall behavior of the loss function for different parameter variations is illustrated in Figure 3.6.

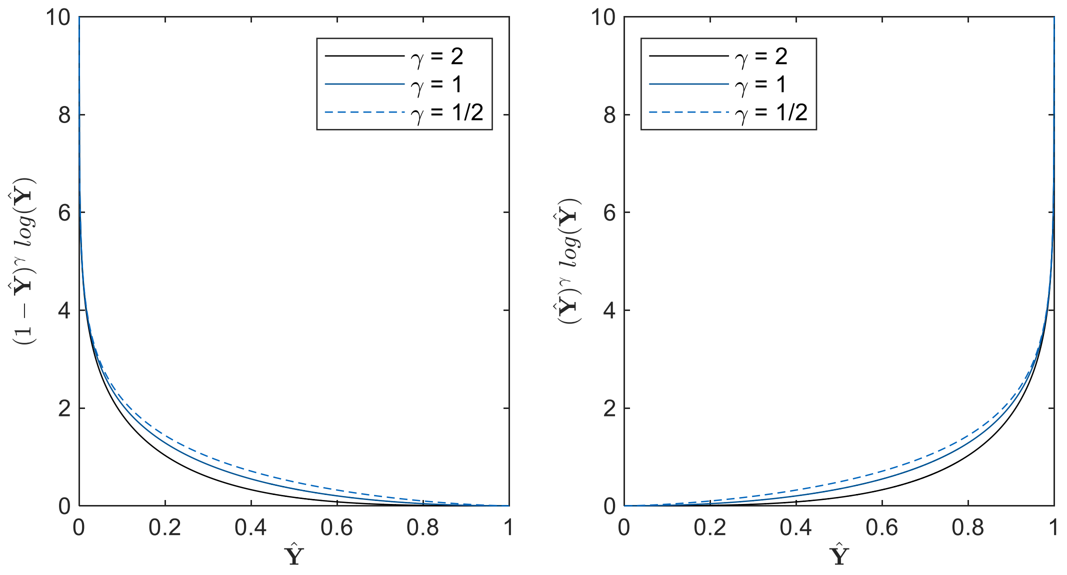


Figure 3.6: Behavior of the loss function according to [48].

3.5 Optimizer

The optimizer describes the method of adjusting the model weights in order to obtain a minimum loss value as a consequence of the model training. The optimization algorithm (the optimizer) builds the core of the gradient-based optimization process to solve

$$\min_{\theta} \mathcal{L}(\mathbf{Y}, \widehat{\mathbf{Y}}) \tag{3.12}$$

as good as possible and can be implemented in different ways [32, pp. 82-86]. Most common optimizers are based on stochastic gradient descent and use different techniques to avoid getting stuck in local minima or to enhance the optimization process. A further explanation of different optimizers and their suitability for different problems can be found in [62, 63].

Within this model the Adam [64] optimizer is chosen as optimization algorithm due to its suitability for optimization problems with many parameters, its robust behavior to the choice of hyperparameters and little demand on system resources [32, p. 309]. While the Adam optimizer is built upon the RMSProp [65] algorithm by additionally utilizing the second-order moments of the gradients to calculate the parameter updates, the learning rate represents the most important parameter of the optimizer [32, p. 308].

The learning rate describes the step size of a single optimization step and is part of the models hyperparameter space. The right choice of the learning rate value is important to ensure a fast degradation of the loss value and a stable training behavior. For this reason, the learning rate is scheduled according to

$$\mathrm{lr}(\mathrm{epoch}) = \mathrm{lr}_0\, \mathrm{dr}^{\frac{\mathrm{epoch}}{\mathrm{ds}}}, \tag{3.13}$$

where lr_0 denotes the initial learning rate, dr the decay rate and ds the decay steps. The final setting of these parameters is determined in a number of experiments and described in Appendix D.

3.6 Evaluation Metric

Not the minimization of the loss value but the optimization of the segmentation quality measured by the evaluation metric is the objective of the model training [32, pp. 275-276]. For that reason, the selection of a suitable evaluation metric is an essential part of the problem definition and subject to the method defined in section 1.2.

The selected evaluation metric must be suited for the given problem as well as the underlying dataset. Given that the utilized nuScenes dataset shows significant imbalances between different classes, as represented in Figure 2.2, metrics like accuracy are inappropriate for the considered problem since they are biased in favor of the majority class [66]. Therefore, the macro averaged F1 score is chosen as main evaluation metric.

The macro averaged F1 score is suitable for the evaluation of models trained on imbalanced datasets and is determined by the arithmetic mean of the class individual F1 scores [21]. The F1 score itself is defined as

$$\mathrm{F1} = 2\,\frac{\mathrm{precision}\ \ \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{3.14}$$

and represents the harmonic mean between precision

$$\mathrm{precision} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}} \tag{3.15}$$

and recall

$$\mathrm{recall} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}}, \tag{3.16}$$

where tp denotes the number of true positives, fp the number of false positives and fn the number of false negatives.

3.7 Experimental Design

The experimental design is subject to the stated research questions of section 1.2 and defines the execution of the model trainings. The goal of the conducted experiments is to provide insights about the influence of the independent variables on the segmentation quality of the model. The independent variables a given by the chosen dataset, the network architecture and the set of hyperparameters. This goal is achieved by varying the independent variables while monitoring the behavior of the model output within a controlled environment.

The controlled environment – which ensures the reliability of the experiments – is subject to a number of different methods addressing the model initialization as well as the execution of the model training. First, the whole process pipeline is created by a defined configuration file including not only the set of hyperparameters but also the definition of the model architecture as well as the dataset composition. Second, the model initialization is seeded to ensure a deterministic behavior of all pseudo random operations and finally, the model training is set to a deterministic execution mode within the defined limitations of [67, p. 12]. Within this environment, the conducted experiments are grouped into three test series based on the method defined in section 1.2 and in association with the stated research questions.

First, a comparison of different state of the art networks – described in subsection 2.4.1 – is given to determine their performance on publicly available datasets. Besides the comparison of the network architectures themselves, the main purpose of this experiment is the provision of comparable results to enhance the interpretability of state of the art results on proprietary datasets. Furthermore, a better understanding for the suitability of the utilized nuScenes [17] dataset is aspired by the provision of these benchmark results.

Second, a series of experiments with different recurrent neural network architectures is performed to discuss the following hypothesis, which is derived from the research question raised in section 1.2.

The utilization of the temporal information enhances the segmentation quality.

To discuss this, three different RNN architectures are evaluated on three different dataset compositions and compared to state of the art feedforward neural networks. These models include the proposed KPLSTM architecture of section 3.3 as well as a KPConv [22] architecture with a complementary LSTM layer within the bridge model and a similar architecture but with a convolutional LSTM [58] layer instead of a standard LSTM [47] layer.

In a third step, different hyperparameter settings are evaluated to determine their influence on the segmentation quality of the model. For that purpose, the KPConv model is chosen since it is not just used within the comparison of different state of the art networks, but it also builds the basis of the utilized RNN architectures. Given this network architecture, the test procedure follows a grid search approach to explore the hyperparameter domain. Therefore, the considered hyperparameters are varied according to a defined set of parameter values, while all other parameters are kept constant to ensure the validity of the experiment. Under consideration of this test procedure, the investigated hyperparameters are chosen to be the kernel size (the number of kernel points) as well as the radius for the neighborhood generation to examine the importance of utilizing the point relationships on the segmentation quality.

4 Results

The results of the experiments, which are executed according to the defined procedure of section 3.7, are represented in the following. The first section 4.1 contains the results of the feedforward neural network architectures, which are trained with focus on their performance on publicly available datasets. The second section 4.2 represents the results of the recurrent neural network architectures in comparison to the results of the feedforward neural network architectures on three different dataset compositions. In the third section 4.3 the results of two hyperparameter studies are represented to a get a better understanding of the model behavior.

All experiments are executed on a remote server equipped with a NVIDIA Tesla V100, an Intel Xeon Skylake processor and 362 GiB of system memory. The model implementation is based on the python programming language (version 3.6.9) and the TensorFlow library version 2.2.0. On this basis, all model trainings are executed according to the method defined in section 3.7 and evaluated by the metric defined in section 3.5.

4.1 Feedforward Neural Network Results

In order to evaluate the performance of state of the art network architectures on publicly available datasets, two feedforward neural networks have been trained on the nuScenes dataset. The PointNet++ architecture is chosen in association to [21] and initialized according to the parameters defined in Table 3.2. Therefore, the PointNet++ architecture consists of three SA modules, three FP modules and an output model with three shared MLP layers as well as two dropout layers. A detailed configuration of the PointNet++ architecture can be found in Appendix C.

The KPConv model is based on the kernel point fully convolutional neural network (KP-FCNN) architecture of [22] and defined in subsection 3.3.1. To maintain the comparability of both network architectures, the KPConv model is also initialized according to the parameters defined in Table 3.2 and further described in Appendix C. The results of both network architectures on the benchmark dataset are represented in Table 4.1 by their macro averaged (total) and class individual F1 scores.

Table 4.1: Results of the feedforward neural network architectures on the benchmark dataset, as defined in Table 3.1. The total F1 score denotes the macro averaged F1 score of all four classes.

Model	Total F1 Score	None F1 Score	Cycle F1 Score	Pedestrian F1 Score	Vehicle F1Score
PointNet++	0.251	0.800	0.006	0.024	0.172
KPConv	0.324	0.916	0.010	0.069	0.300

4.2 Recurrent Neural Network Results

The second research question is subject to the utilization of the temporal domain and its effects on the segmentation quality of the model. To discuss this, several experiments have been conducted to evaluate the segmentation quality of three RNN architectures trained on three different compositions of the nuScenes dataset, as defined in section 3.1. The network architectures include two KPConv models with an additional recurrent bridge model, one of which is equipped with a standard LSTM layer according to [47] and another with a ConvLSTM layer according to [58]. Besides that, the proposed KPLSTM model architecture defined in subsection 3.3.2 is evaluated and its results are represented in Table 4.2.

Table 4.2: Results of the recurrent neural network architectures on the dataset compositions, defined in section 3.1, in comparison to state of the art feed forward neural networks. All models are evaluated according to the macro averaged F1 score.

Encoder Model	Bridge Model	Benchmark dataset	Dynamic vehicle dataset	Pedestrian dataset
PointNet++	-	0.251	0.470	0.382
KPConv	-	**0.324**	0.480	**0.505**
KPConv	LSTM	0.276	0.480	0.379
KPConv	ConvLSTM	0.312	**0.730**	0.475
KPLSTM	-	0.249	0.644	0.377

4.3 Hyperparameter Optimization

The results of the hyperparameter studies are obtained on the KPConv model architecture and are subject to the third research question on the effects of different hyperparameters on the segmentation quality of the model. The first study examines the effects of the kernel size on the model results and is executed by varying the number of kernel points while monitoring the evaluation metric. The results of this experiment are visualized in Figure 4.1 as a plot of the macro averaged F1 score over the kernel size.

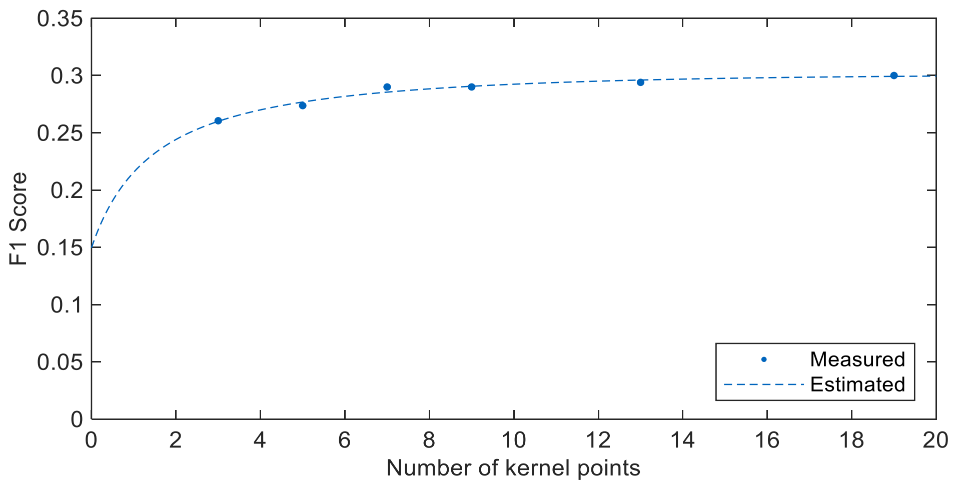


Figure 4.1: Results of the kernel point hyperparameter study. The plot represents the achieved macro averaged F1 score for different numbers of kernel points.

The second hyperparameter study is executed in the same way, by modifying just the radius parameter of the encoder modules and maintaining all other settings. In consideration of the experimental design defined in section 3.7, the radii of the encoder modules are adjusted according to

$$\mathrm{r}_i = 2^i \mathrm{r}_0, \tag{4.1}$$

where r_i denotes the radius parameter of the i^{th} encoder module and r_0 the radius of the initial encoder stage. To examine the effects of the radius configuration on the segmentation quality of the model, the initial radius r_0 is varied while measuring the macro averaged F1 score of the model, as represented in Figure 4.2.

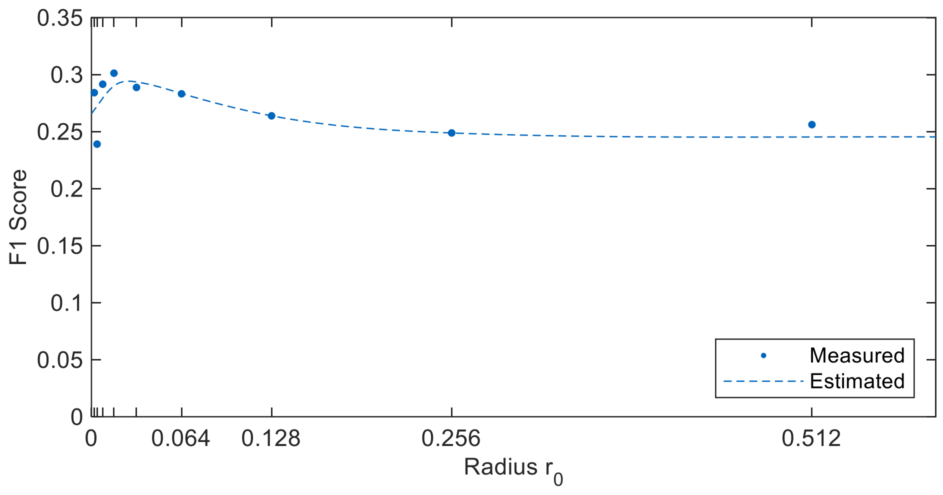


Figure 4.2: Results of the radius hyperparameter study represented by the macro averaged F1 score over the initial encoder radius. The radii of the remaining encoder modules are determined according to $\mathrm{r}_i = 2^i \mathrm{r}_0$ for $i = 0, \dots, 2$.

5 Discussion

The following chapter discusses the results of the thesis, the underlying model architecture, and the applied experimental design. The chapter is structured according to the method defined in section 1.2 and with regards to the stated research questions. Therefore, the experimental design is discussed first – in section 5.1 – in consideration of the quality criteria of empirical research. Building upon this, the utilized dataset is discussed in section 5.2 and the performance of the feedforward neural networks on the nuScenes dataset is discussed in section 5.3. In section 5.4 different recurrent neural network architectures, including the proposed network architecture of section 3.3, are discussed and compared to the current state of the art. Finally, the results of the hyperparameter studies are discussed in section 5.5.

5.1 Experimental Design

The experimental design is chosen in order to provide a defined training and evaluation procedure that ensures the comparability of different results. Under these conditions, the conducted experiments are subject to the three quality criteria of empirical research given by the objectivity, reliability and validity of the experiment [68, p. 138].

The objectivity of the experiment describes the independency of the results from the observer and can be further split into the objectivity of the procedure, the test evaluation and the interpretation of the results [69, p. 70]. The first aspect of the objectivity is ensured by a consistent execution of the experiments according to a defined configuration file which specifies the entire process pipeline. Considering the objectivity of the test evaluation, an appropriate evaluation metric is defined in section 3.5 to be able to compare different model results by a standardized method. However, the interpretation of the results is subject to the observer themself and can limit the objectivity of this aspect.

Reliability is a criterion for the consistency of a measurement and can be interpreted as the independency of the results from the number of experiment executions [68, p. 138]. To fulfill this quality criterion of empirical research, several measures are taken to create a controlled environment for the experiment execution, as described in section 3.7. However, the parallelization of the model training limits the reliability of the experiment due to the non-deterministic behavior of specific operations, as defined in [67, p. 12]. This limitation mostly affects network architectures build upon convolution or sampling operations and will especially influence model trainings with a high number of training epochs. However, the observation of this behavior is limited to the RNN experiments discussed in section 5.4.

The validity of the experiment, which describes the correspondence of the observation with the property to be observed, is subject to the monitoring process and the chosen evaluation metric. According to this the evaluation metric should be able to represent the segmentation quality of the model, which is the aspired optimization goal of the project. However, the macro averaged F1 score does not allow to draw conclusions about the class individual or even pointwise segmentation quality of the model but represents a global quality value. To overcome this limitation, one could either use the class individual F1 scores or the confusion matrix of the training, but even that one would not be able to represent the confidence of the model's prediction on the individual classes. On the contrary, an objective comparison of different experiments requires the provision of a defined comparative figure limiting the suitability of non-scalar quality metrics. For both that reason and its suitability to evaluate models trained on imbalanced datasets, the macro averaged F1 score is used as main evaluation metric, as described in section 3.5. However, the suitability of different evaluation metrics is an ongoing topic of research and further discussed in [32, 66, 70].

5.2 The nuScenes Dataset

The discussion about the performance of state of the art network architectures on the nuScenes dataset has to take the dataset itself into account. Since the provided dataset is the only source of information and highly related to the achievable segmentation quality, a discussion of the same is indispensable [59, pp. 48-58].

Information theory implies that all information originates from the elements themselves or the relationship between elements [71]. For that reason, the amount of information carried by the element (point) as well as the number of available elements defines the entirety of available information. However, the amount of information carried by an element is not just subject to its quantity but also to its quality, hance both aspects must be considered.

The information content of a single point is defined by its radar channels, which are specified by their value range and resolution – listed in Table 2.2. Within this context, the restriction of the rcs channel to a value range between -5.0 and 63.5 dBm^2 represents a significant limitation, due to the importance of the rcs value on the differentiability between classes, as shown by [48]. This is especially true if one considers that all radar channels are derived from the four base measurement values given by the distance, the relative velocity, the azimuth angle and the rcs value of the radar detection [72].

Besides the information content of the elements themselves, the number of elements is essential to the overall available information [71]. With this in mind, two aspects are essential to evaluate the value of the available information. First, the number of elements per object is important, which defines the informational content of the object. Figure 5.1 gives some insights about the available number of radar points per object of the utilized classes of the benchmark dataset and shows that they are significantly lower than within comparable datasets like the one used by [1]. The characteristic statistical values of the other datasets can be found in the Appendix B.

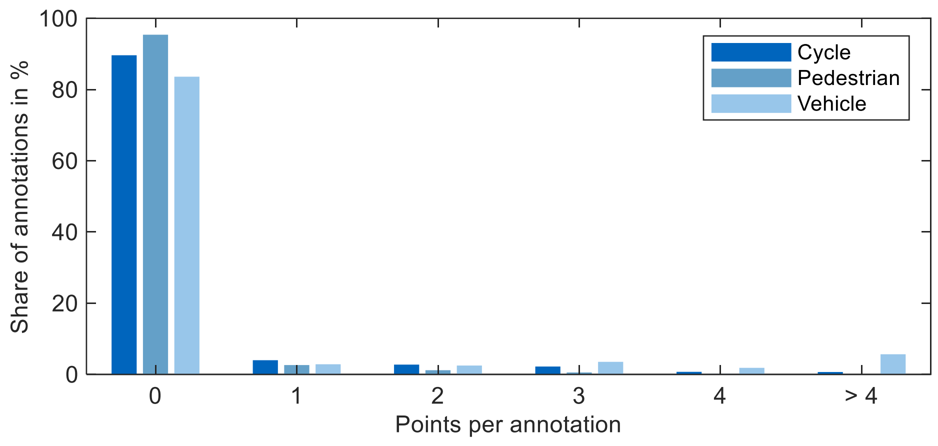


Figure 5.1: Share of class annotations (bounding boxes) with a certain number of associated radar points of the benchmark dataset.

The second aspect concerns the distribution of the elements across different classes, which is represented in Figure 5.2. The given distributions show huge imbalances between classes, which has two effects on the model training. First, the model performance on the minority classes is negatively affected due to the comparatively low amount of available information [73]. Even if the required amount of data to successfully train a model is an ongoing research topic of statistical power analysis, statistics show that more data generally leads to better results [73–76].

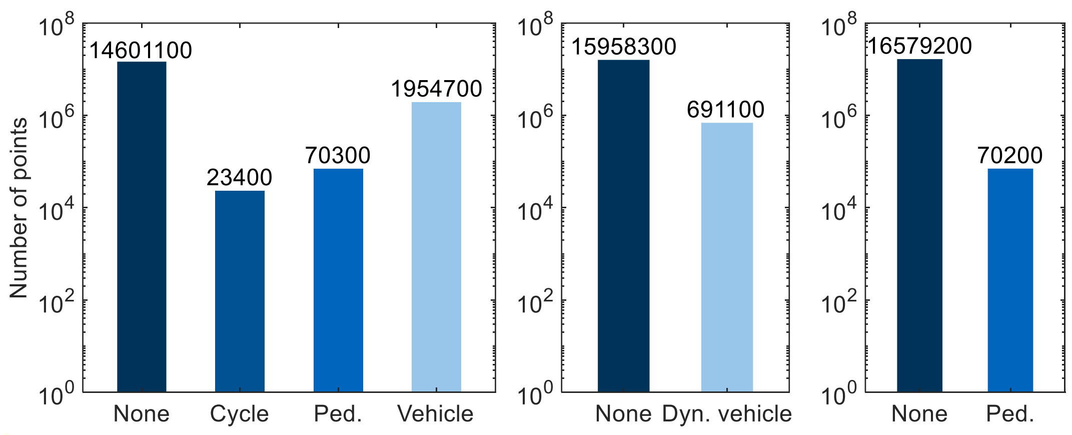


Figure 5.2: Class distributions of the three different dataset compositions on a logarithmic scale. The left figure represents the benchmark dataset, the middle one the dynamic vehicle dataset and the right one the pedestrian dataset.

Second, the imbalanced class distribution directly affects the behavior of the model training since elements of the majority class occur more often in the calculation of the loss value. For that reason, most traditional machine learning approaches are biased towards the majority class [66, 77, 78]. To counteract this problem, a class weighted loss function based on the focal loss function [60] is applied. However, the class imbalances still represent one of the greatest challenges during the hyperparameter tuning and the tendency towards the majority class can still be observed within the training results, as represented in Table 4.1.

5.3 Feedforward Neural Networks

The purpose of this section is to discuss the feedforward neural network architectures, as defined in subsection 3.3.1. The results of the PointNet++ [23] architecture are discussed first and compared to the current state of the art. Beyond that, the results of the KPConv [22] architecture are discussed in a second step and compared to those of the PointNet++ architecture.

The PointNet++ model reaches a macro averaged F1 score of 0.25 on the benchmark dataset and shows a noticeable tendency towards the majority class. This tendency is represented in Table 4.1 and corresponds with the results of [21, 42]. The observation of this tendency follows the general assumption that most traditional machine learning approaches are biased towards the majority class and is further discussed in [48].

Besides that, the model achieves the best performance on the vehicle class followed by the pedestrian and cycle class. Comparing these results to the number of available points in Figure 5.2 it is noticeable that the model performance on these classes corresponds to the number of available points. This observations is in line with the current state of the art [21, 42] and indicates that more data is required to successfully train a model on the underrepresented classes as well as the demand on further research on the topic of handling imbalanced datasets.

To put these results into context, the segmentation quality of the PointNet++ model is compared to the current state of the art. Schumann et al. [21] reached a macro averaged F1 score of 0.74 on a proprietary dataset, while Cennamo et al. [42] achieved a score of 0.67 with the same model architecture but on a different proprietary dataset. Applying a similar PointNet++ architecture to the nuScenes dataset, a macro averaged F1 score of 0.25 can be achieved. The results of this experiment are compared to the those of Schumann et al. in Figure 5.3 and discussed in the following.

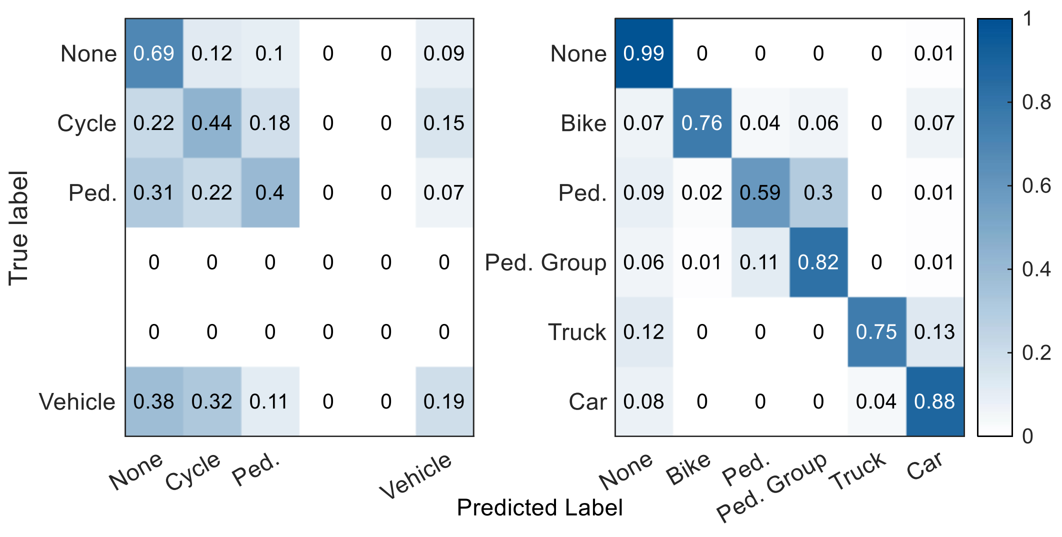


Figure 5.3: Comparison of the results of the PointNet++ architecture on the benchmark dataset on the left and the results of Schumann et al. [21] on the right. Note that the number of classes and the composition of the classes is not equal.

Observation shows that both reference models reach a higher evaluation score and therefore a better segmentation quality. The difference in the segmentation quality can be subject to two root causes. First, differences in the model architectures cannot be excluded, even if all models are based on the PointNet++ architecture. This is due to the fact that both network configurations are not publicly available, which makes an exact replication of the network architecture impossible. However, this issue is considered to be subordinated since all major model parameters are replicated and a variation of the remaining hyperparameters does not improve the segmentation quality.

Besides possible deviations in the model architecture, the key difference between the three models is the utilized dataset. The model of SCHUMANN et al. is built upon a dataset with up to 3072 radar points per keyframe, whereas the nuScenes dataset is limited to a maximum of 625 points per keyframe [21]. In comparison to that, CENNAMO et al. uses an input point cloud with a maximum of 1200 points per keyframe [42]. In addition to that, not only the total number of available points is higher than within the nuScenes dataset but also the point density. Therefore, the informational content of the different datasets is the key difference between the three models and results suggest that the underlying dataset has a huge impact on the achievable segmentation quality of the model.

Beyond the current state of the art, a model based on the KPConv [22] architecture is trained on the same datasets as the PointNet++ architecture. This model achieves a macro averaged F1 score of 0.32 on the benchmark dataset and therefore surpasses the results of the PointNet++ architecture. In addition to that, the KPConv-based architecture achieves not only a higher macroscopic evaluation score but also exceeds the class individual results of the PointNet++ architecture. A comparison between both model results is given in Table 4.1 and illustrated in Figure 5.4.

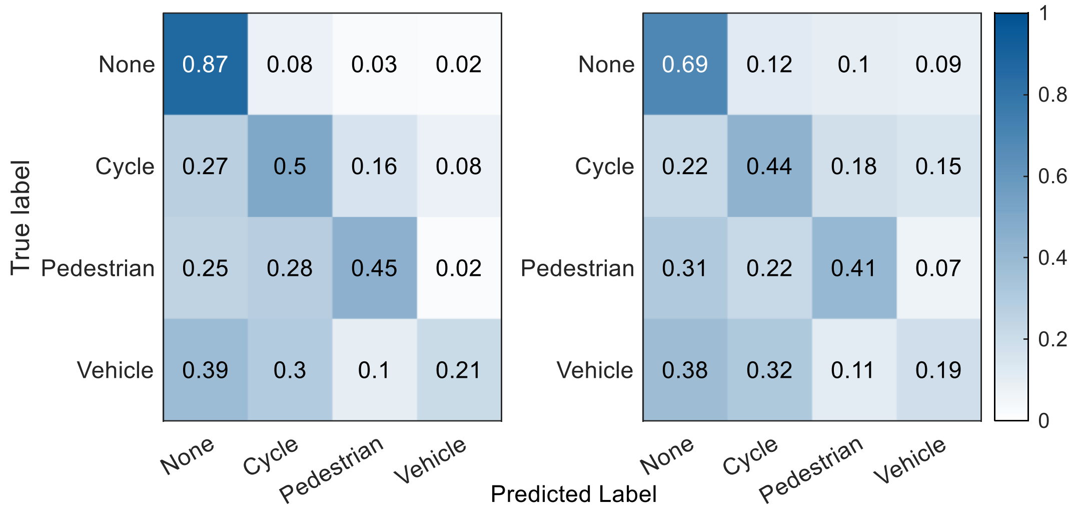


Figure 5.4: Comparison of the results of the KPConv architecture (on the left) with the results of the PointNet++ architecture (on the right).

Apart from these results, the KPConv model outperforms the PointNet++ architecture also in the pedestrian and dynamic vehicle dataset and achieves the best results across all tested network architectures except for the dynamic vehicle dataset. The higher evaluation score of the KPConv architecture in comparison to the PointNet++ architecture can be explained by the fact that the PointNet++ architecture represents a particular configuration of the KPConv model and is therefore included in the possible range of solutions.

In addition to the advantages on the segmentation quality, the KPConv-based architecture shows a faster and more stable degradation of the loss value than the PointNet++ architecture. This behavior is shown in Figure 5.5, where the training of the KPConv model reaches loss value convergence already after 400 epochs, while the PointNet++ model training took over 1200 epochs. Moreover, the training of the PointNet++ model shows jumps within the loss value progression, which indicates an unusable training process. These observations corresponds with the argument of THOMAS et al. [22] that a pointwise MLP operation makes the convolution operator more complex and the convergence of the network harder.

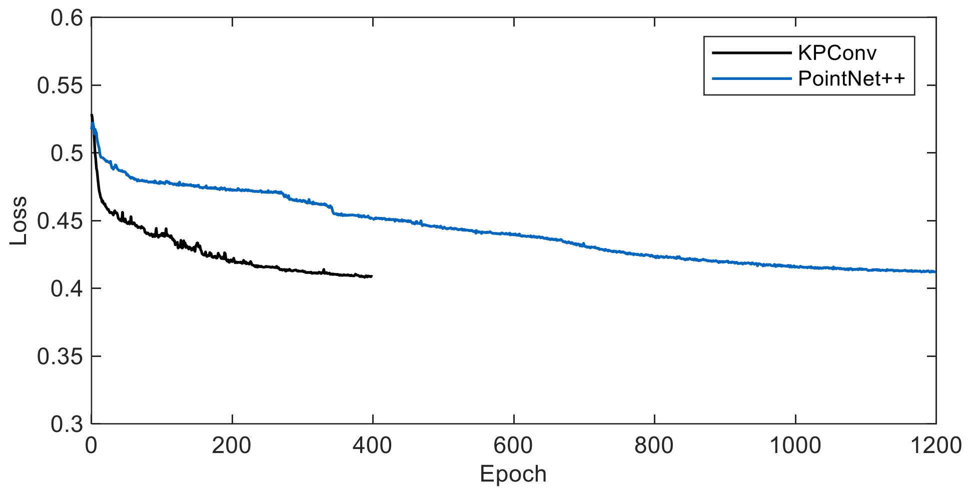


Figure 5.5: Comparison of the training loss value progression of the KPConv architecture and the PointNet++ architecture on the benchmark dataset.

In summary, the results of SCHUMANN et al. [21] cannot be reproduced and it is assumed that this is a result of the limited informational content of the nuScenes dataset in comparison their proprietary dataset. However, a new model architecture based on the KPConv [22] architecture is proposed, which outperforms the PointNet++ model in all tested dataset compositions. Nevertheless, a final valuation of the KPConv architecture is difficult since it has not yet been applied to any other automotive radar data related problems.

5.4 Recurrent Neural Networks

To discuss the effects of utilizing the temporal information of the provided radar data, three different recurrent neural network architectures have been defined in subsection 3.3.2. These networks extend the previously discussed feedforward neural networks and include two architectures with a recurrent bridge model as well as the proposed KPLSTM architecture. The results of these model trainings are discussed in the following to evaluate the influence of the temporal domain on the segmentation quality.

In a first step the KPConv architecture is extended by the introduction of a standard LSTM layer within the model's bridge module. However, this extension of the network architecture had a negative effect on the segmentation quality as a result. This outcome can be observed for every dataset composition except for the dynamic vehicle dataset, where the same macro averaged F1 score as the KPConv-based architecture can be reached, as represented in Table 4.2.

The negative effects of introducing a standard LSTM layer can be caused by two aspects of the model architecture. First, a higher number of model parameters increases the model complexity and therefore the likelihood of overfitting [8, pp. 104-105]. Second, the required reduction of the data dimensions limits the capability of the model to represent the actual feature space. This issue is caused by the implementation of a standard LSTM cell, which supports only three-dimensional input data. However, the considered radar data has four dimensions as described in section 2.3, which is why the point and feature dimension is combined to enable an utilization of the LSTM layer.

Besides these two aspects, the LSTM model is not invariant to input permutations due to the fully connected layers within the gate units of the LSTM cell. Furthermore, the network cell is not robust to correlation-preserving transformations and lacks the ability to handle spatiotemporal data, as shown by [58]. Conclusively, the model architecture does not satisfy the model requirements and leads to unsatisfying segmentation results, while increasing the demand on system resources.

Building upon this, a model architecture with a ConvLSTM [58] bridge model has been trained and evaluated according to the method defined in section 3.7. The model achieves state of the art results on the dynamic vehicle dataset, as represented in Figure 5.6, while performing slightly worse than the KPConv-based model architecture on the benchmark and pedestrian dataset, as listed in Table 4.2. Besides that, the model training is characterized by a fast degradation of the loss value but a strong tendency towards overfitting. This behavior is represented in Figure 5.6 and discussed in the subsequent paragraphs.

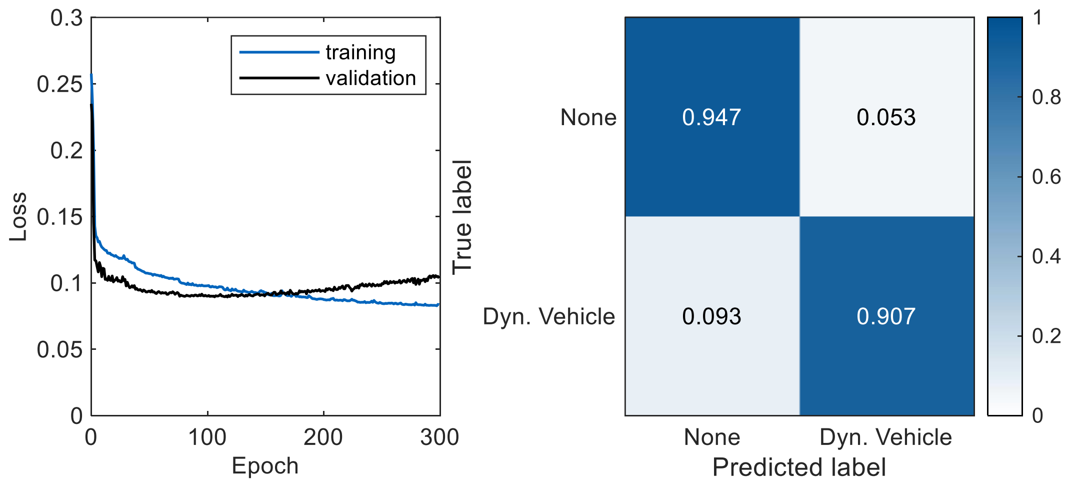


Figure 5.6: Results and training behavior of the model architecture with ConvLSTM bridge model, represented by its loss value progression on the left and its confusion matrix of the 101th epoch on the right.

As already stated, the model's tendency towards overfitting can be subject to a high number of model parameters but also a result of too little training data, missing regularization of the model weights or an inappropriate configuration [8, pp. 105-110]. However, the introduction of weight regularization terms or the addition of dropout layers has not improved the segmentation quality of the network. Nevertheless, due to high number of hyperparameters and the limited computational resources it was not possible to explore the whole model design space, which is why further research is required to generalize the results of the model architecture.

Besides that, the ConvLSTM cell can only process input data with a fixed number of input points. This property limits the possible implementations of a ConvLSTM layer to the bridge model architecture, where the number of input points is defined by the encoder configuration. Moreover, the ConvLSTM architecture is not invariant to input permutations in case a kernel size greater than one is chosen, which limits its suitability for point cloud input data. However, it is worth mentioning that the applied radar sensor sorts the radar points by their distance and therefore applies an intrinsic order to them [18]. For this reason and due to the fact that the input data of the ConvLSTM layer is highly encoded (reduced to a small number of points) and thus limits the number of possible permutations, no negative effects on the segmentation quality can be observed.

To overcome the limitations of the ConvLSTM architecture and to utilize the temporal information on different encoding stages, a novel model architecture – called KPLSTM – has been proposed in subsection 3.3.2. KPLSTM can process input data of varying input sizes and is able to capture dynamically changing relations among points over time due to its origin in the PointRNN architecture [58]. Furthermore, the KPLSTM architecture is invariant to input permutations and has the ability to capture relationships among neighboring points since it is built upon a KPConv [22] backbone. Moreover, the network architecture is robust to correlation-preserving transformations due to the utilized convolution operation, as shown by [22]. Therefore, all requirements on a model architecture capable of processing point cloud data and considering temporal information are met by the KPLSTM model.

Applying this network architecture to the nuScenes dataset, the KPLSTM model reaches a macro averaged F1 score of 0.25 on the benchmark dataset and therefore a comparable result to the PointNet++ architecture. The best result is achieved on the dynamic vehicle dataset and an example prediction out on this dataset is given in Figure 5.7. However, both the KPConv-based model architecture as well as the ConvLSTM model surpasses the performance of the KPLSTM model on different dataset compositions, as shown in Table 4.2.

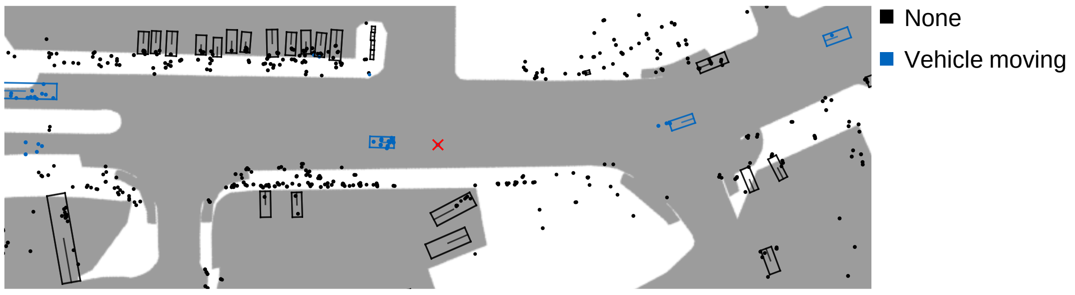


Figure 5.7: Example prediction of the KPLSTM model to differentiate moving vehicles from the surroundings. The point colors represent the actual model prediction, whereas the bounding boxes represent the ground truth data. This result is achieved on dynamic vehicle dataset.

This result is subject to several characteristics of the KPLSTM model as well as the model's training behavior. First, the model training is orders of magnitudes slower than the training of the other network architectures, which is a result of the missing cuDNN support for the custom built KPLSTM cell. As a result, the training time per epoch, with about 1200 s for the KPLSTM model, is significantly higher than the 90 s of the KPConv model. For that reason, a training time limitation has to be set to restrict the demand on computational resources even if the loss value still degrades at a constant rate, as shown in Figure 5.8. This observation suggests that the full potential of the KPLSTM model is not yet used and further improvements on the segmentation quality of the model can be expected [8, p. 104].

In addition to that, vanishing gradients can be observed during the model training leading to a degradation of layer updates and an unstable training behavior. Even though the root cause analysis of this effect is difficult, HOCHREITER et al. [47] identified an unimpeded flow of information through the cell as crucial property to avoid vanishing gradients. However, this unimpeded flow of information is not ensured by the PointRNN [46] implementation on which the KPLSTM cell is built upon. Moreover, the KPLSTM cell is not embedded into a ResNet architecture, like the KPConv model, which addresses vanishing gradients by the introduction of a skiplink connection [56]. In consequence, the problem of vanishing gradients is not considered within the model architecture and further research is required to overcome this problem.

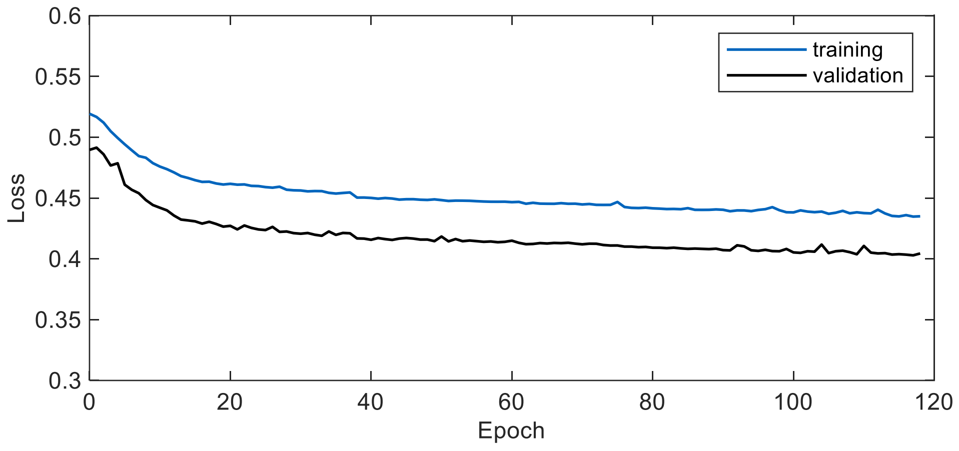


Figure 5.8: Loss value progression of the KPLSTM model training on the benchmark dataset.

In summary, the KPConv-based model architecture outperforms other state of the art network architectures and achieves the best segmentation quality on the benchmark dataset as well as the pedestrian dataset, measured by the macro averaged F1 score. In contrast, all recurrent neural network architectures surpass the segmentation quality of the feedforward neural network architectures on the dynamic vehicle dataset. This suggests that RNN architectures are more suitable to distinguish between dynamic objects, while feedforward neural networks generalize better across multiple classes. However, to provide evidence on this conjecture more research and the utilization of different datasets is required.

5.5 Hyperparameter Optimization

To discuss the third research question raised in section 1.2 and to explore the effects of utilizing the information of the point relationships on the segmentation quality, two hyperparameter studies are executed. The considered hyperparameters are subject to the KPConv kernel and therefore directly linked to the ability of capturing relationships among neighboring points. The KPConv model is chosen as object of investigation because of its utilization within the feedforward neural network architectures and the fact that it builds the backbone architecture of the KPLSTM model.

The first hyperparameter study evaluates the effects of the kernel size on the segmentation quality of the model by varying the number of kernel points. The results show that more kernel points generally lead to better results, while the improvement stagnates for higher point counts, as represented in Figure 4.1. This observation corresponds to the findings of [22] and suggests that the optimum number of kernel points is at approximately nine points. More kernel points increase the risk of overfitting and raise the demand on system resources, while at the same time the benefit of utilizing more kernel points decreases with an increasing number of points.

The second hyperparameter study examines the influence of the radius of the local neighborhoods on the segmentation quality of the model. The radius defines the area around the center points which is considered to group local neighborhoods and corresponds to the spatial size of the convolution kernel. The model reaches a maximum in segmentation quality for a radius of 0.016 (8 m), while showing a larger increase for smaller radii and reaches an equilibrium for greater radii, as represented in Figure 4.2. This result corresponds to [23] and can be explained by the fact that a small radius limits the number of reachable points (and therefore the number of usable point relationships), whereas larger radii tend to cover the entire space and therefore cannot reach more points with ever increasing radius.

6 Conclusion

A summary of the development process and the obtained results is given in section 6.1 to conclude the stated research questions of section 1.2. Building upon these results, an outlook is given in section 6.2 to motivate further research on the topic of machine learning-based radar point cloud segmentation.

6.1 Summary

The utilization of radar sensors to perceive the environment around the vehicle is driven by the demand of autonomous vehicles to ensure their functionality within difficult traffic scenarios and challenging environmental conditions [79]. This demand on the functional safety of road vehicles is highly related to reliable sensor information and redundant system architectures [2]. Therefore, it is indispensable to utilize different sensor technologies and develop independent system architectures for individual sensor systems. Thus, the utilization of radar point cloud data to perceive the surrounding environment is the objective of this thesis.

To reach this goal an artificial neural network is developed to semantically segment the radar point cloud by assigning a class label to every radar point. The development process of this model is characterized by the selection of a suitable dataset, the development of the network architecture and the optimization of the model [8]. Associated with these modeling steps are three research questions on the performance of state of the art network architectures on publicly available datasets, the influence of utilizing the temporal domain on the segmentation quality of the model as well as the effects of different hyperparameters.

Following this method, the selection of a dataset is drawn from a comparison of different publicly available datasets for autonomous driving. As a result, the nuScenes [17] dataset is chosen as data basis since it is the only publicly available dataset with annotated radar data and the information about consecutive keyframes. However, within the included radar data less than 12 % of the objects have even one radar point associated to them, which limits the overall available information. In addition to that, the nuScenes dataset is characterized by an imbalanced class distribution making it difficult to successfully train a model on the provided radar data [48].

The discussion of the first research question is subject to two feedforward neural network architectures trained on the nuScenes dataset. Based on the results of the PointNet++ [23] architecture in comparison to those of SCHUMANN [21] and CENNAMO [42], it is suggested that the performance of state of the art network architectures is highly dependent on the underlying dataset. Furthermore, it can be shown that the KPConv [22] model architecture outperforms other network architectures trained on the nuScenes dataset. Nevertheless, the segmentation quality of state of the art network architectures trained on proprietary datasets cannot be met.

Associated with the second research question is the development of a novel recurrent neural network architecture to evaluate the impact of the utilization of the temporal domain on the segmentation quality. This novel network architecture is derived from the PointRNN [46] model by replacing the gate units of the LSTM cell with modified KPConv layers. Alongside this model, two other RNN architectures are proposed by deploying a LSTM [47] and ConvLSTM [58] layer within the autoencoder structure of the KPConv network. The results of these models suggest that RNN architectures are more suitable to distinguish between dynamic objects, while feedforward neural networks are better suited for multi-class segmentation problems.

Based on the KPConv model architecture two hyperparameter studies were conducted to evaluate their effects on the segmentation quality of the model. In this context, the size of the convolution kernel as well as the radius of the neighborhoods have been varied to investigate the importance of utilizing point relationships. The results of these studies imply that a larger convolution kernel generally leads to better results, whereas an ideal neighborhood radius can be found to achieve the best results.

In summary, this thesis shows that the performance of state of the art network architectures depends on the underlying dataset and demonstrates the vulnerabilities of the nuScenes dataset. Furthermore, it can be implied that the chosen network architecture has to be suited for the aspired segmentation task, since RNN architectures perform better on the distinction of dynamic classes, while feedforward neural networks generalize better across multiple classes. Finally, all three research questions have been discussed on the experimental results, while new questions occurred during the project that are open for research.

6.2 Outlook

Despite the fact that the stated research questions have been discussed in accordance with the associated project goal, new questions arose from the experimental results. These open research questions involve the utilized dataset, the applied network architecture as well as the deployed optimization process.

The utilized dataset represented one of the greatest challenges during the development process because of its imbalanced class distribution and the sparsity of the provided radar data. To overcome this problem one could either acquire a new dataset or implement data augmentation methods to enhance the suitability of the available data. Such methods include the modification of existing data samples as well as the generation of synthetic data to increase the number of radar points and compensate the class imbalances [36, 80]. Nevertheless, more suitable datasets for automotive radar data are required and further research on the topic of data augmentation is needed.

Even if multiple feedforward and recurrent neural network architectures have been successfully applied to semantically segment automotive radar data, additional network architectures are to be considered. The extension of the KPConv architecture to utilize deformable convolution kernels has shown good results on non-automotive segmentation tasks and therefore represents a promising enhancement to the existing model architecture [22]. Besides that, good results have been achieved with the ConvLSTM architecture, which is why an implementation of the same should be considered within the encoder structure. In addition to that, a compensation of the diverging point locations by their relative velocities could enhance the performance of the proposed KPLSTM architecture, as shown by [1]. Furthermore, a modification of the KPLSTM cell has to be considered in order to tackle the vanishing gradient problem and comply with the idea of [47] to enable an unimpeded flow of information through the cell. However, the most promising approach seems to be the one of CENNAMO et al. [42] who uses several small networks for specific segmentation tasks and aggregates their predictions by an attention mechanism to solve a multiclass segmentation problem on automotive radar data. Similar to this approach, feedforward neural networks and RNNs could be combined to achieve an overall better segmentation quality.

Finally, a suitable hyperparameter tuning has to be applied to efficiently explore the model parameter space. This is especially true because of the huge parameter space of the utilized network architectures, the high computational costs of RNN architectures and their sensitivity to small parameter changes. Under this consideration one could either use a Bayesian optimization [81] process or genetic algorithms [82] to optimize the hyperparameters. However, the determination of a suitable optimization process is subject to future research to utilize the full potential of the provided network architectures for machine learning-based radar point cloud segmentation.

List of Figures

List of Tables

Bibliography

[1] O. Schumann, J. Lombacher, M. Hahn, C. Wohler, and J. Dickmann, “Scene Understanding With Automotive Radar,” *IEEE Trans. Intell. Veh,* vol. 5, no. 2, p. 188–203, 2020.

[2] *ISO 26262-3: Road vehicles — Functional safety: — Part 3: Concept phase*, International Organization for Standardization, Geneva, Switzerland, 2018.

[3] F. de Ponte Müller, “Survey on Ranging Sensors and Cooperative Techniques for Relative Positioning of Vehicles,” *Sensors (Basel, Switzerland)*, vol. 17, no. 2, 2017.

[4] M. Bijelic *et al,* “Seeing Through Fog Without Seeing Fog: Deep Multimodal Sensor Fusion in Unseen Adverse Weather,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, p. 11679–11689.

[5] J. Betz, M. Lienkamp, and B. Lohmann, “Artificial Intelligence in Automotive Technology: Introduction: Artificial Intelligence,”: *Technical University of Munich*, Garching, Munich, 2019.

[6] *ISO 26262: Road Vehicles — Functional Safety*, International Organization for Standardization, Geneva, Switzerland, 2011.

[7] A. Powell: *Brown bridge and vehicles.* Accessed: Mar. 20 2020.

[8] F. Chollet: *Deep learning with Python*, Shelter Island, NY, Manning Publications Co, 2018.

[9] M. Cordts *et al,* “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016): Las Vegas, Nevada, USA, 27-30 June 2016*, Piscataway, NJ: IEEE, 2016, p. 3213–3223.

[10] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ: IEEE, 2012, p. 3354–3361.

[11] A. Patil, S. Malla, H. Gang, and Y.-T. Chen, “The H3D Dataset for Full-Surround 3D Multi-Object Detection and Tracking in Crowded Urban Scenes,” in *2019 International Conference on Robotics and Automation (ICRA)*, [Piscataway, NJ]: IEEE, 2019, p. 9552–9557.

[12] Y. Choi *et al,* “KAIST Multi-Spectral Day/Night Data Set for Autonomous and Assisted Driving,” *IEEE Trans. Intell. Transport. Syst,* vol. 19, no. 3, p. 934–948, 2018.

[13] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, “1 year, 1000 km: The Oxford RobotCar dataset,” *The International Journal of Robotics Research*, vol. 36, no. 1, p. 3–15, 2017.

[14] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner, “The Oxford Radar RobotCar Dataset: A Radar Extension to the Oxford RobotCar Dataset,” Sep. 2019.

[15] F. E. Nowruzi *et al,* “Deep Open Space Segmentation using Automotive Radar,” Mar. 2020.

[16] M. Meyer and G. Kuschk, “Automotive Radar Dataset for Deep Learning Based 3D Object Detection,” in *2019 16th European Radar Conference: 2-4 October 2019, Paris, France*, [Louvain-la-Neuve, Belgium]: EuMA, 2019, p. 129–132.

[17] H. Caesar *et al,* “nuScenes: A multimodal dataset for autonomous driving,” Mar. 2019.

[18] Continental Engineering Services GmbH, Ed, “Standardized ARS Interface: Technical Documentation. ARS 404-21 (Entry), ARS 408-21 (Premium),” May. 2017.

[19] H. Caesar, "Additional Sensor Information on the nuScenes Dataset", private communication, Feb. 2020.

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” Dec. 2016.

[21] O. Schumann, M. Hahn, J. Dickmann, and C. Wohler, “Semantic Segmentation on Radar Point Clouds,” in *21st International Conference on Information Fusion (FUSION)*, Piscataway, NJ: IEEE, 2018, p. 2179–2186.

[22] H. Thomas *et al,* “KPConv: Flexible and Deformable Convolution for Point Clouds,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*: IEEE, Oct. 2019 - Nov. 2019, p. 6410–6419.

[23] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” Jun. 2017.

[24] A. A. Fraenkel, Y. Bar-Hillel, A. Lévy, and D. van Dalen: *Foundations of set theory,* 2nd ed, Amsterdam, New York, Elsevier Science, 1973.

[25] R. Dedekind, “Was sind und was sollen die Zahlen?,” in vol. 3, *Gesammelte mathematische Werke*, R. Fricke, E. Noether, and Ö. Ore, Eds. 6th ed, Braunschweig: Vieweg, 1930, p. 335–391.

[26] J. Lombacher, M. Hahn, J. Dickmann, and C. Wohler, “Object classification in radar using ensemble methods,” in *2017 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, Piscataway, NJ: IEEE, 2017, p. 87–90.

[27] O. Schumann, C. Wohler, M. Hahn, and J. Dickmann, “Comparison of random forest and long short-term memory network performances in classification tasks using radar,” in *2017 Symposium on Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, Piscataway, NJ: IEEE, 2017, p. 1–6.

[28] S. Heuel and H. Rohling, “Pedestrian recognition based on 24 GHz radar sensors,” in *11th International Radar Symposium (IRS)*, Piscataway, NJ: IEEE, 2010, p. 1–6.

[29] S. Heuel and H. Rohling, “Two-stage pedestrian classification in automotive radar systems,” in *12th International Radar Symposium (IRS)*, Meckenheim: DCM Druck, 2011, p. 477–484.

[30] S. Heuel and H. Rohling, “Pedestrian classification in automotive radar systems,” in *13th International Radar Symposium (IRS), 2012*, Piscataway, NJ: IEEE, 2012, p. 39–44.

[31] T. D. Bufler and R. M. Narayanan, “Radar classification of indoor targets using support vector machines,” *IET Radar, Sonar & Navigation*, vol. 10, no. 8, p. 1468–1476, 2016.

[32] I. Goodfellow, Y. Bengio, and A. Courville: *Deep learning*, Cambridge, Massachusetts, London, England, MIT Press, 2016.

[33] G. Dong and H. Liu: *Feature engineering for machine learning and data analytics*, Boca Raton, FL, CRC Press/Taylor & Francis Group, 2018.

[34] N. Scheiner, N. Appenrodt, J. Dickmann, and B. Sick, “Radar-based Feature Design and Multiclass Classification for Road User Recognition,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*: IEEE, Jun. 2018 - Jun. 2018, p. 779–786.

[35] J. Lombacher, K. Laudt, M. Hahn, J. Dickmann, and C. Wohler, “Semantic radar grids,” in *28th IEEE Intelligent Vehicles Symposium*, Piscataway, NJ: IEEE, 2017, p. 1170–1175.

[36] Y. Zhou and O. Tuzel, “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,” Nov. 2017.

[37] B. Graham and L. van der Maaten, “Submanifold Sparse Convolutional Networks,” Jun. 2017.

[38] C. R. Qi *et al,* “Volumetric and Multi-view CNNs for Object Classification on 3D Data,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, [S.l.]: IEEE, Jun. 2016 - Jun. 2016, p. 5648–5656.

[39] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric Deep Learning: Going beyond Euclidean data,” *IEEE Signal Process. Mag,* vol. 34, no. 4, p. 18–42, 2017.

[40] D. Raposo *et al,* “Discovering objects and their relations from entangled scene representations,” Feb. 2017.

[41] A. Danzer, T. Griebel, M. Bach, and K. Dietmayer, “2D Car Detection in Radar Data with PointNets,” in *The 2019 IEEE Intelligent Transportation Systems Conference - ITSC*, Piscataway, NJ: IEEE, 2019, p. 61–66.

[42] A. Cennamo, F. Kaestner, and A. Kummert, “Leveraging Radar Features to Improve Point Clouds Segmentation with Neural Networks,” in *Proceedings of the International Neural Networks Society, Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference*, L. Iliadis, P. P. Angelov, C. Jayne, and E. Pimenidis, Eds, Cham: Springer International Publishing, 2020, p. 119–131.

[43] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, “SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters,” Mar. 2018.

[44] F. Groh, P. Wieschollek, and H. P. A. Lensch, “Flex-Convolution,” in *Computer Vision – ACCV 2018*, Cham: Springer International Publishing, 2019, p. 105–122.

[45] Y. Li *et al,* “PointCNN: Convolution On $\mathcal{X}$-Transformed Points,” Jan. 2018.

[46] H. Fan and Y. Yang, “PointRNN: Point Recurrent Neural Network for Moving Point Cloud Processing,” Oct. 2019.

[47] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” (eng), *Neural computation*, vol. 9, no. 8, p. 1735–1780, 1997.

[48] F. Fent, “Machine Learning based Object Classification with Automotive Radar Sensors,” Term Paper, Chair of Automotive Technology: *Technical University of Munich*, Munich, 2020.

[49] TensorFlow: *TFRecord and tf.train.Example.* Accessed: Sep. 24 2020.

[50] C. Zhang, M. Fiore, I. Murray, and P. Patras, “CloudLSTM: A Recurrent Neural Model for Spatiotemporal Point-cloud Stream Forecasting,” Jul. 2019.

[51] X. Shi *et al,* “Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds.: Curran Associates, Inc, 2017, p. 5617–5627.

[52] A. Zheng and A. Casari: *Feature engineering for machine learning: Principles and techniques for data scientists / Alice Zheng and Amanda Casari*, Sebastopol, CA, O'Reilly, 2018.

[53] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, “Geodesic Convolutional Neural Networks on Riemannian Manifolds,” in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*: IEEE, Dec. 2015 - Dec. 2015, p. 832–840.

[54] N. Verma, E. Boyer, and J. Verbeek, “FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*: IEEE, Jun. 2018 - Jun. 2018, p. 2598–2606.

[55] Y. Wang *et al,* “Dynamic Graph CNN for Learning on Point Clouds,” Jan. 2018.

[56] 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): Las Vegas, Nevada, USA, 27-30 June 2016*, Piscataway, NJ: IEEE, 2016, p. 770–778.

[57] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, p. 1929–1958, 2014.

[58] X. Shi *et al,* “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting,” Jun. 2015.

[59] C. M. Bishop: *Pattern recognition and machine learning,* 11th ed, New York, Springer, 2013.

[60] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” Aug. 2017.

[61] TensorFlow: *Classification on imbalanced data.* Accessed: Sep. 27 2020.

[62] S. Ruder, “An overview of gradient descent optimization algorithms,” Sep. 2016.

[63] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A Survey of Optimization Methods From a Machine Learning Perspective,” (eng), *IEEE transactions on cybernetics*, vol. 50, no. 8, p. 3668–3681, 2020.

[64] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Dec. 2014.

[65] G. Hinton, “Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude,”: *Coursera*, 2012.

[66] P. Soda, “A Hybrid Approach Handling Imbalanced Datasets,” in *Image analysis and processing - ICIAP 2009: 15th international conference, Vietri sul Mare, Italy, September 8 - 11, 2009 ; proceedings*, Berlin: Springer, 2009, p. 209–218.

[67] Nvidia, “cuDNN Developer's Guide,”: *Nvidia*, 2019.

[68] M. Eisend and A. Kuß: *Grundlagen empirischer Forschung*, Wiesbaden, Springer Fachmedien Wiesbaden, 2017.

[69] P. Sedlmeier and F. Renkewitz: *Forschungsmethoden und Statistik für Psychologen und Sozialwissenschaftler,* 3rd ed, Hallbergmoos, Pearson, 2018.

[70] D.M.W. Powers, “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation,” *Journal of Machine Learning Technologies*, vol. 2, no. 1, p. 37–63, 2011.

[71] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, no. 3, p. 379–423, 1948.

[72] A. Ludloff: *Praxiswissen Radar und Radarsignalverarbeitung*, Wiesbaden, Vieweg+Teubner Verlag, 2002.

[73] R. L. Figueroa, Q. Zeng-Treitler, S. Kandula, and L. H. Ngo, “Predicting sample size required for classification performance,” (eng), *BMC medical informatics and decision making*, vol. 12, pp. 8, 2012.

[74] J. Cohen: *Statistical Power Analysis for the Behavioral Sciences,* 2nd ed, Hoboken, Taylor and Francis, 1988.

[75] C. J. Adcock, “Sample size determination: a review,” *J Royal Statistical Soc D*, vol. 46, no. 2, p. 261–283, 1997.

[76] R. V. Lenth, “Some Practical Guidelines for Effective Sample Size Determination,” *The American Statistician*, vol. 55, no. 3, p. 187–193, 2001.

[77] J. M. Johnson and T. M. Khoshgoftaar, “Survey on deep learning with class imbalance,” *J Big Data*, vol. 6, no. 1, pp. 3, 2019.

[78] H. Partamian, Y. Rizk, and M. Awad, “Barricaded Boundary Minority Oversampling LS-SVM for a Biased Binary Classification,” in *Lecture Notes in Artificial Intelligence*, vol. 11198, *Discovery Science: 21st International Conference, DS 2018, Limassol, Cyprus, October 29-31, 2018, Proceedings*, L. Soldatova, J. Vanschoren, G. Papadopoulos, and M. Ceci, Eds, Cham: Springer International Publishing; Imprint: Springer, 2018, p. 18–32.

[79] *ISO 21448: Road vehicles — Safety of the intended functionality*, International Organization for Standardization, Geneva, Switzerland, 2019.

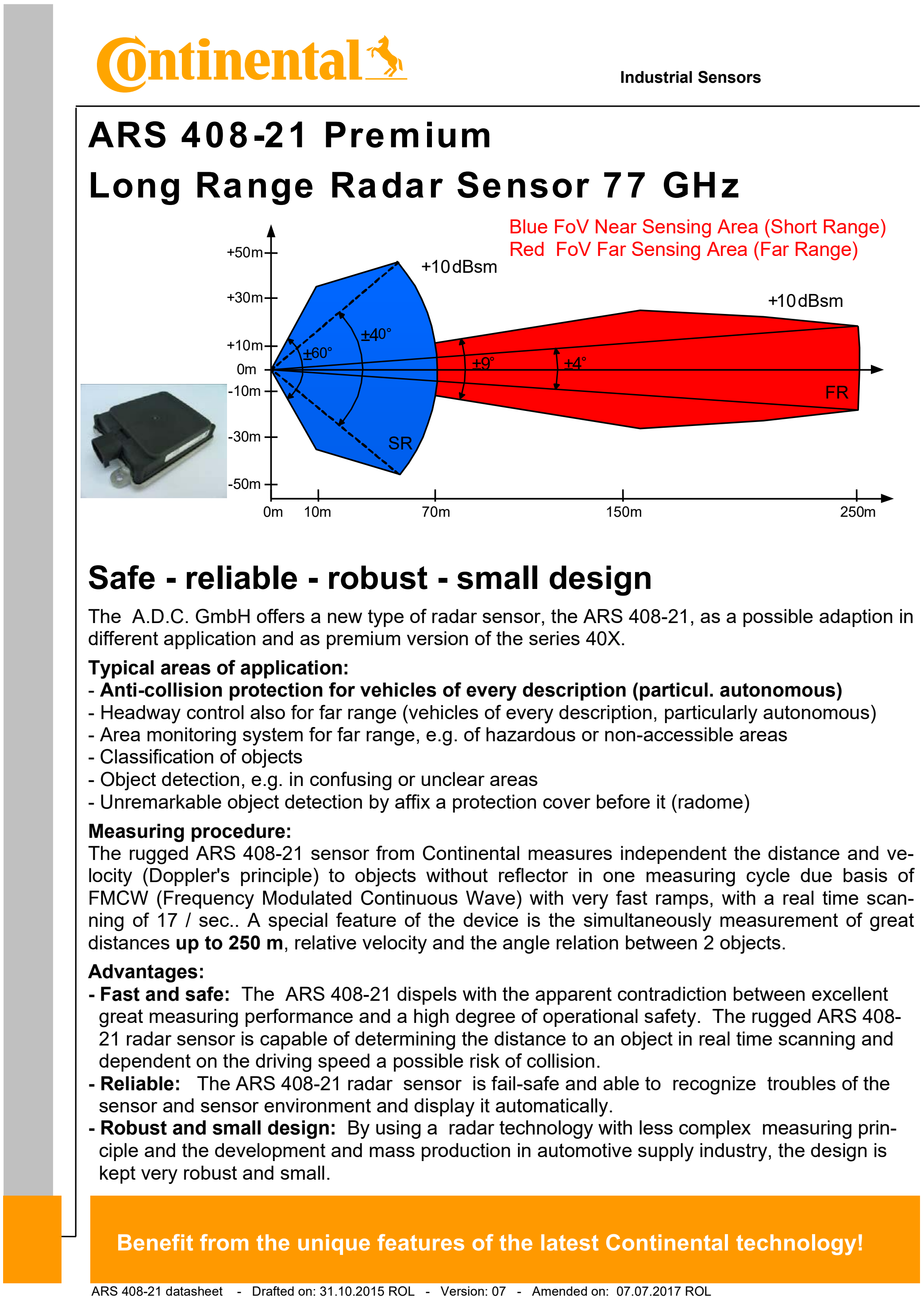
[80] Y. Yan, Y. Mao, and B. Li, “SECOND: Sparsely Embedded Convolutional Detection,” (eng), *Sensors (Basel, Switzerland)*, vol. 18, no. 10, 2018.

[81] P. I. Frazier, “A Tutorial on Bayesian Optimization,” Jul. 2018.

[82] J.-H. Han, D.-J. Choi, S.-U. Park, and S.-K. Hong, “Hyperparameter Optimization Using a Genetic Algorithm Considering Verification Time in a Convolutional Neural Network,” *J. Electr. Eng. Technol,* vol. 15, no. 2, p. 721–726, 2020.

Appendix

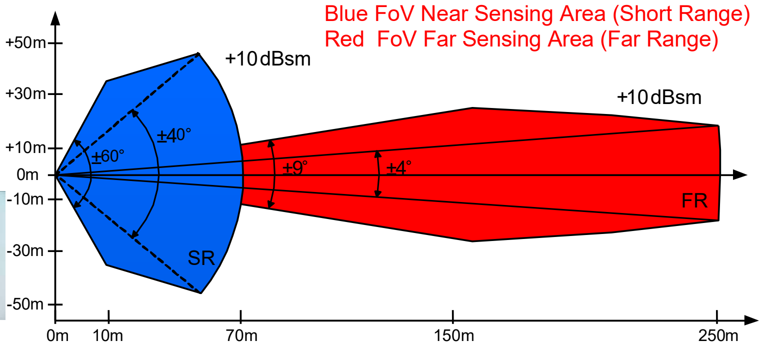
A Radar Sensor Specifications



Continental — Industrial Sensors

ARS 408-21 Premium

Long Range Radar Sensor 77 GHz


Blue FoV Near Sensing Area (Short Range)
Red FoV Far Sensing Area (Far Range)
+10dBsm
+10dBsm
+50m
+30m
+10m
0m
-10m
-30m
-50m
±60°
±40°
±9°
±4°
SR
FR
0m
10m
70m
150m
250m

Safe - reliable - robust - small design

The A.D.C. GmbH offers a new type of radar sensor, the ARS 408-21, as a possible adaption in different application and as premium version of the series 40X.

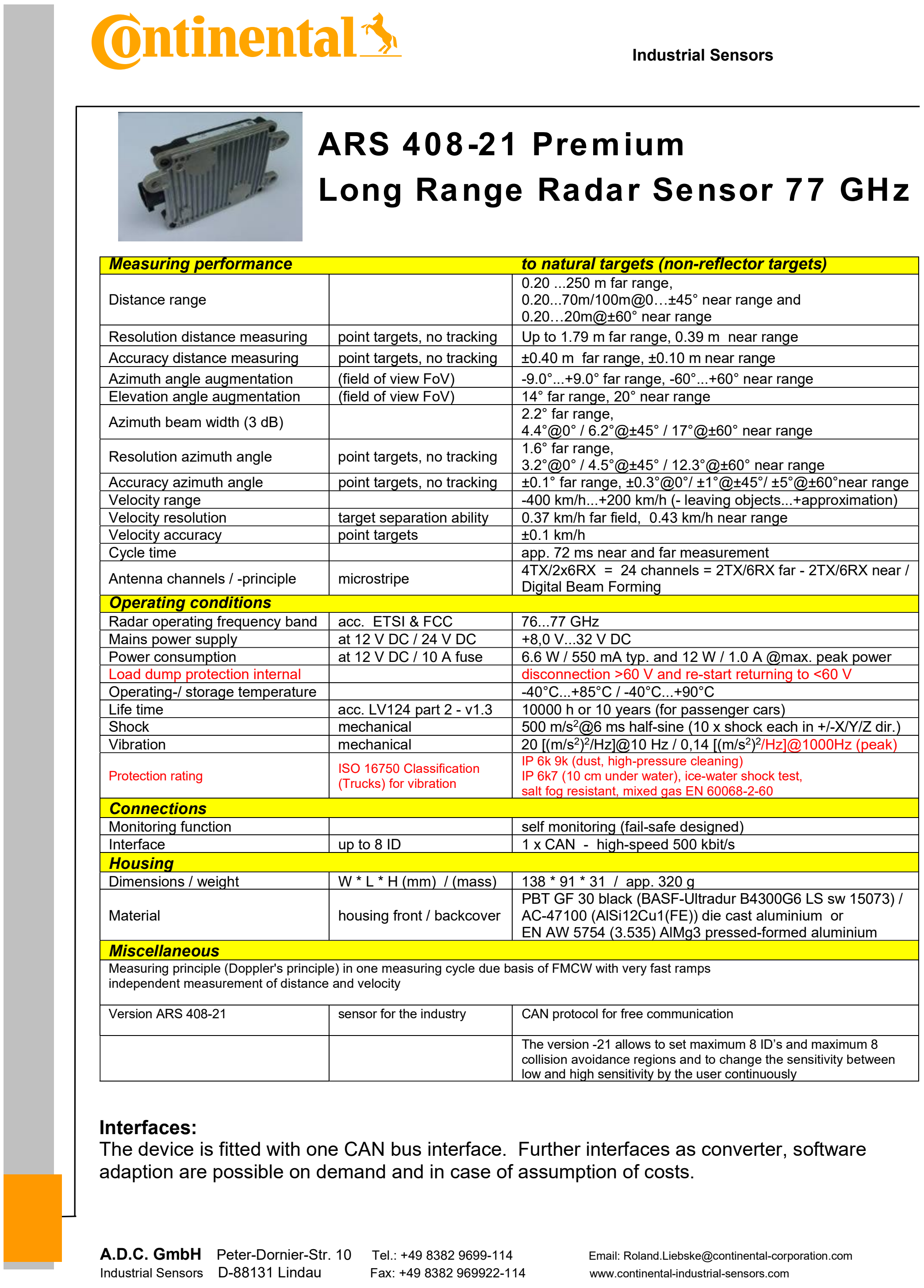
Typical areas of application:
- **Anti-collision protection for vehicles of every description (particul. autonomous)**
- Headway control also for far range (vehicles of every description, particularly autonomous)
- Area monitoring system for far range, e.g. of hazardous or non-accessible areas
- Classification of objects
- Object detection, e.g. in confusing or unclear areas
- Unremarkable object detection by affix a protection cover before it (radome)

Measuring procedure:
The rugged ARS 408-21 sensor from Continental measures independent the distance and velocity (Doppler's principle) to objects without reflector in one measuring cycle due basis of FMCW (Frequency Modulated Continuous Wave) with very fast ramps, with a real time scanning of 17 / sec.. A special feature of the device is the simultaneously measurement of great distances **up to 250 m**, relative velocity and the angle relation between 2 objects.

Advantages:
- **Fast and safe:** The ARS 408-21 dispels with the apparent contradiction between excellent great measuring performance and a high degree of operational safety. The rugged ARS 408-21 radar sensor is capable of determining the distance to an object in real time scanning and dependent on the driving speed a possible risk of collision.
- **Reliable:** The ARS 408-21 radar sensor is fail-safe and able to recognize troubles of the sensor and sensor environment and display it automatically.
- **Robust and small design:** By using a radar technology with less complex measuring principle and the development and mass production in automotive supply industry, the design is kept very robust and small.

Benefit from the unique features of the latest Continental technology!

ARS 408-21 datasheet - Drafted on: 31.10.2015 ROL - Version: 07 - Amended on: 07.07.2017 ROL



Industrial Sensors

ARS 408-21 Premium

Long Range Radar Sensor 77 GHz

Measuring performance		***to natural targets (non-reflector targets)***
Distance range		0.20 ...250 m far range, 0.20...70m/100m@0…±45° near range and 0.20…20m@±60° near range
Resolution distance measuring	point targets, no tracking	Up to 1.79 m far range, 0.39 m near range
Accuracy distance measuring	point targets, no tracking	±0.40 m far range, ±0.10 m near range
Azimuth angle augmentation	(field of view FoV)	-9.0°...+9.0° far range, -60°...+60° near range
Elevation angle augmentation	(field of view FoV)	14° far range, 20° near range
Azimuth beam width (3 dB)		2.2° far range, 4.4°@0° / 6.2°@±45° / 17°@±60° near range
Resolution azimuth angle	point targets, no tracking	1.6° far range, 3.2°@0° / 4.5°@±45° / 12.3°@±60° near range
Accuracy azimuth angle	point targets, no tracking	±0.1° far range, ±0.3°@0°/ ±1°@±45°/ ±5°@±60°near range
Velocity range		-400 km/h...+200 km/h (- leaving objects...+approximation)
Velocity resolution	target separation ability	0.37 km/h far field, 0.43 km/h near range
Velocity accuracy	point targets	±0.1 km/h
Cycle time		app. 72 ms near and far measurement
Antenna channels / -principle	microstripe	4TX/2x6RX = 24 channels = 2TX/6RX far - 2TX/6RX near / Digital Beam Forming
Operating conditions		
Radar operating frequency band	acc. ETSI & FCC	76...77 GHz
Mains power supply	at 12 V DC / 24 V DC	+8,0 V...32 V DC
Power consumption	at 12 V DC / 10 A fuse	6.6 W / 550 mA typ. and 12 W / 1.0 A @max. peak power
Load dump protection internal		disconnection >60 V and re-start returning to <60 V
Operating-/ storage temperature		-40°C...+85°C / -40°C...+90°C
Life time	acc. LV124 part 2 - v1.3	10000 h or 10 years (for passenger cars)
Shock	mechanical	500 m/s^2@6 ms half-sine (10 x shock each in +/-X/Y/Z dir.)
Vibration	mechanical	20 [(m/s^2)^2/Hz]@10 Hz / 0,14 [(m/s^2)^2/Hz]@1000Hz (peak)
Protection rating	ISO 16750 Classification (Trucks) for vibration	IP 6k 9k (dust, high-pressure cleaning) IP 6k7 (10 cm under water), ice-water shock test, salt fog resistant, mixed gas EN 60068-2-60
Connections		
Monitoring function		self monitoring (fail-safe designed)
Interface	up to 8 ID	1 x CAN - high-speed 500 kbit/s
Housing		
Dimensions / weight	W * L * H (mm) / (mass)	138 * 91 * 31 / app. 320 g
Material	housing front / backcover	PBT GF 30 black (BASF-Ultradur B4300G6 LS sw 15073) / AC-47100 (AlSi12Cu1(FE)) die cast aluminium or EN AW 5754 (3.535) AlMg3 pressed-formed aluminium
Miscellaneous		
Measuring principle (Doppler's principle) in one measuring cycle due basis of FMCW with very fast ramps independent measurement of distance and velocity		
Version ARS 408-21	sensor for the industry	CAN protocol for free communication
		The version -21 allows to set maximum 8 ID's and maximum 8 collision avoidance regions and to change the sensitivity between low and high sensitivity by the user continuously

Interfaces:
The device is fitted with one CAN bus interface. Further interfaces as converter, software adaption are possible on demand and in case of assumption of costs.

A.D.C. GmbH Peter-Dornier-Str. 10 Tel.: +49 8382 9699-114 Email: Roland.Liebske@continental-corporation.com
Industrial Sensors D-88131 Lindau Fax: +49 8382 969922-114 www.continental-industrial-sensors.com

Figure A.1: Radar sensor specifications of the Continental ARS 408-21 Premium. This sensor is used to record the radar data of the nuScenes dataset.

The following Table A.1 provides detailed descriptions of the radar channels of the ARS 408-21 Premium radar sensor. As already mentioned in section 2.2 the radar sensor provides eight general and seven quality information values per point. In addition to these 15 values, nuScenes provides three additional values consisting of the vertical coordinate z, the relative velocity in longitudinal direction compensated by the vehicle ego motion vx_comp and its equivalent in lateral direction vy_comp. This means, that the three additional channels of the nuScenes dataset are not part of the specified radar interface and the corresponding descriptions are provided by nuTonomy.

Table A.1 Description of the signal channels of the ARS 408-21 radar sensor according to [18].

Channel Id	Signal channel	Channel description
0	x	Longitudinal (x) coordinate
1	y	Lateral (y) coordinate
2	z	Vertical (z) coordinate
3	dyn_prob	Dynamic property of cluster to indicate if it is moving or not
4	id	Cluster number
5	rcs	Radar cross section
6	vx	Relative velocity in longitudinal direction (x)
7	vy	Relative velocity in lateral direction (y)
8	vx_comp	Relative velocity in longitudinal direction (x) compensated by the vehicle ego motion in longitudinal direction
9	vy_comp	Relative velocity in lateral direction (y) compensated by the vehicle ego motion in lateral direction
10	is_quality_valid	Flag whether a radar cluster is valid or not
11	ambig_state	0x0: invalid 0x1: ambiguous 0x2: staggered ramp 0x3: unambiguous 0x4: stationary candidates
12	x_rms	Standard deviation of longitudinal distance
13	y_rms	Standard deviation of lateral distance

Channel Id	Signal channel	Channel description
14	invalide_state	0x00: Valid
		0x01: Invalid due to low RCS
		0x02: Invalid due to near-field artefact
		0x03: Invalid far range Cluster because not confirmed in near range
		0x04: Valid Cluster with low RCS
		0x05: reserved
		0x06: Invalid Cluster due to high mirror probability
		0x07: Invalid because outside sensor field of view
		0x08: Valid Cluster with azimuth correction due to elevation
		0x09: Valid Cluster with high child probability
		0x0A: Valid Cluster with high probability of being a 50 deg artefact
		0x0B: Valid Cluster but no local maximum
		0x0C: Valid Cluster with high artefact probability
		0x0D: reserved
		0x0E: Invalid Cluster because it is a harmonic
		0x0F: Valid Cluster above 95 m in near range
		0x10: Valid Cluster with high multi-target probability
		0x11: Valid Cluster with suspicious angle
15	pdh0	0x0: invalid
		0x1: <25%
		0x2: <50%
		0x3: <75%
		0x4: <90%
		0x5: <99%
		0x6: <99.9%
		0x7: <=100%
16	vx_rms	Standard deviation of longitudinal relative velocity
17	vy_rms	Standard deviation of lateral relative velocity

B nuScenes Radar Data Analysis

Table B.1 represents the number of radar points per class of the three dataset compositions, as defined in section 3.1. The differance of the benchmark and pedestrian dataset within the pedestrian class can be expressed by the different assignment of the animal category, as represented in Table 3.1.

Table B.1 Number of radar points per class for the three dataset compositions of the nuScenes dataset.

Class	Benchmark dataset	Dyn.vehicle dataset	Pedestrian dataset
None	14601085	15958322	16579199
Cycle	23368	-	-
Pedestrian	70254	-	70185
Vehicle	1954677	-	-
Vehicle moving	-	691062	-

To evaluate the information content of the provided radar data of the nuScenes dataset and to illustrate the sparsity of this data, an representation of the associated number of radar points per annotation (bounding box) across the three classes of benchmark dataset is given in Table B.2.

Table B.2 Number of radar points per class for the three classes of the benchmark dataset.

Class	Median	75th percentile	95th percentile	99th percentile
Cycle	0	0	0	2
Pedestrian	0	0	2	4
Vehicle	0	0	5	10

C Model Specifications

In order to ensure the reproducibility of the results and to give more details on the network configuration, the following appendix lists the key hyperparameter settings of the different network architectures. First, the hyperparameter settings of the PointNet++ encoder modules are given in Table C.1. In addition to that all MLP layers within the encoder modules use no bias values but a rectified linear unit (ReLU) as activation function.

Table C.1: Key hyperparameter settings of the three encoder modules of the PointNet++ network architecture.

Encoder module	Number of center points	Max. number of neighboring points	Neighborhood radius r_0	Number of filters
0	512	16	0.001	16, 16, 32
1	128	8	0.002	32, 32, 64
2	16	4	0.004	64, 64, 128

Similar to the PointNet++ encoder modules, the KPConv encoder is initialized with the same hyperparameter values except for the neighborhood radius r_0. The neighborhood radius is chosen in accordance with the results of the hyperparameter study of section 4.3.

Table C.2: Key hyperparameter settings of the three encoder modules of the KPConv network architecture.

Encoder module	Number of center points	Max. number of neighboring points	Neighborhood radius r_0	Number of kernel points	Number of filters
0	512	16	0.016	9	16, 16, 32
1	128	8	0.032	9	32, 32, 64
2	16	4	0.064	9	64, 64, 128

The same applies to the hyperparameter setting of the KPLSTM model, which is already discussed in subsection 3.3.2. However, it is worth mentioning that the underlying model structure is not built upon a ResNet configuration but in accordance with the architecture of Figure 3.4.

Table C.3: Key hyperparameter settings of the three encoder modules of the KPLSTM network architecture.

Encoder module	Number of center points	Max. number of neighboring points	Neighborhood radius r_0	Number of kernel points	Number of filters
0	512	16	0.016	9	16, 16, 32
1	128	8	0.032	9	32, 32, 64
2	16	4	0.064	9	64, 64, 128

The RNN architecture with a recurrent LSTM bridge model is initialized in accordance with the settings defined in Table C.4. The remaining submodels of the network architecture are implemented just like the KPConv architecture.

Table C.4: Key hyperparameter settings of the LSTM bridge module.

Units	Use bias	Activation	Recurrent activation	Kernel initializer	Recurrent initializer
131	False	tanh	sigmoid	ones	zeros

The ConvLSTM-based RNN applies the same principles as the LSTM-based version of the network and is therefore built upon a KPConv network configuration with an additional bridge model, as defined in Table D.5.

Table C.5: Key hyperparameter settings of the ConvLSTM bridge module.

Units	Kernel size	Use bias	Activation	Recurrent activation	Kernel initializer	Recurrent initializer
131	(3, 1)	False	tanh	hard sigmoid	ones	zeros

The decoder model of all networks architectures is implemented as PointNet++ decoder with three FP modules, as discussed in section 3.3. The key hyperparameter settings of the FP modules are listed in Table C.6.

Table C.6: Key hyperparameter settings of the three decoder modules.

Decoder module	Use bias	Activation	Number of filters
0	False	ReLU	128, 128
1	False	ReLU	128, 64
2	False	ReLU	64, 64, 64

The output model of all network architectures is realized by a set of three consecutive shared MLP layers and two intermediate dropout layers. The settings of these layers are listed in Table C.7, whereas the filters of the last layer are chosen in accordance with the number of classes of the individual dataset compositions.

Table C.7: Layer configuration and key hyperparameter settings of the output module.

Layer	Dropout rate	Use bias	Activation	Number of filters
MLP	-	False	ReLU	32
Dropout	0.5	-	-	-
MLP	-	False	ReLU	16
Dropout	0.5	-	-	-
MLP	-	False	softmax	Number of classes

D Ablation Studies

The ablation studies are not strictly bound the experimental design of section 3.7 and serve the purpose of providing more insights about the behavior of the tested model architectures. In consideration of this, the behavior of the PointNet++ architecture for different settings of the learning rate decay dr is represented in Table D.1. The learning rate decay describes the degradation of the learning rate over training epochs and is subject to the gradient-based optimization process as described in section 3.5.

The provided results of the PointNet++ architecture are subject to the experimental determination of the learning rate settings and are also used for the initialization of other model trainings. It can be shown that a decay rate of 0.9 leads to the best segmentation quality with a macro averaged F1 score of 0.251, as represented in Table D.1. For that reason, a learning rate decay of 0.9 is chosen as benchmark configuration for all tested model architectures.

Table D.1: Results of the PointNet++ model architecture for different settings of the learning rate decay on the benchmark dataset.

Learning rate decay dr	Total F1 Score	None F1 Score	Cycle F1 Score	Pedestrian F1 Score	Vehicle F1Score
0.35	0.240	**0.961**	0	0	0
0.7	0.210	0.680	**0.011**	**0.073**	0.076
0.8	0.229	0.910	0.008	0	0
0.9	**0.251**	0.800	0.006	0.024	**0.172**
0.95	0.240	**0.961**	0	0	0